A Dynamic System Testbed to Facilitate Controller and Estimator Designs and Its Application to Neoclassical Controllers

Justin James Kennah

Marquette University

A DYNAMIC SYSTEM TESTBED TO FACILITATE CONTROLLER AND ESTIMATOR DESIGNS AND ITS APPLICATION TO NEOCLASSICAL CONTROLLERS

by

Justin James Kennah

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

May 2021

ABSTRACT

A DYNAMIC SYSTEM TESTBED TO FACILITATE CONTROLLER AND ESTIMATOR DESIGNS AND ITS APPLICATION TO NEOCLASSICAL CONTROLLERS

Justin James Kennah

Marquette University, 2021

This thesis proposes a testbed capable of validating experimental controller designs. Because it is undesirable to implement unvalidated controller designs on physical hardware, the need for an effective testbed arises. For the testbed to be useful, it should be easy to use and contain all the hardware and software necessary to implement a controller and analyze its response. To accomplish these goals, a testbed is created by joining the National Instruments ELVIS III board with the Quanser Controls Board.

Three controllers are implemented to evaluate the usefulness of the testbed. The controllers are designed using the neoclassical design technique. The neoclassical technique is an experimental design methodology that, until now, has never been implemented on a physical system. The neoclassical technique uses principles from both modern and classical approaches to control to match the transfer function of a system to an optimal transfer function; thus, simplifying the design process. Neoclassical controllers are designed in continuous time, discrete time, and discrete time with a state estimator.

A PID controller is designed and used to evaluate the usefulness of the neoclassical design technique. PID controllers have been around for many years and are a highly common method of control. They have been studied extensively and there are several techniques available for controller design. Because the PID is well-known, it provides a good standard to evaluate the neoclassical technique against.

A design and performance analysis is then conducted after all controllers have been implemented. The testbed is evaluated on how well it performed as well as the difficulty required to implement the individual controller designs. The performance of the neoclassical controllers is also considered. A comparison is made between the neoclassical controllers and PID controller to determine if the neoclassical method improves efficiency of controller design.

It will be shown that while the testbed analyzed in this thesis may be useful as an instructional aid, it is not particularly useful in validating experimental controllers.

TABLE OF CONTENTS

List of Figures

List of Tables

ACKNOWLEDGMENTS

Justin James Kennah

I would like to first, and foremost, express my gratitude to my wife, Kim. Her love, support, and patience made this work possible. I want to thank my children, Vincent and Jackson, for giving me inspiration. I would like to thank my parents, James and Joan, for always believing in me.

Additionally, I would like to thank my advisor, Dr. Susan Schneider, for her help and guidance throughout this process. I would also like to thank the members of my thesis committee, Dr. Edwin Yaz and Dr. Jennifer Bonniwell, for providing direction and guidance.

Chapter 1: Introduction

1.1 History of Control Systems and Theory

Control systems are very common in the world today. They can be found in simple devices that control the speed of a motor to far more complex systems that direct the trajectory of a rocket ship. Essentially, the objective of a control system is to automatically control the response of a dynamic system. For the process to be automatic, feedback is used to allow the system to monitor its output. The system uses the information provided by feedback to make adjustments to the system until the desired output is achieved.

It may be surprising to learn that feedback control systems have been around for thousands of years. The first known application of a feedback-based control system can be traced back to a water clock developed by the ancient Egyptians around the third century B.C.E. [2]. Control systems would continue to advance in sophistication, and in application of use, through the years. By the mid 1700's control systems had found their way into industry. The first known industrial application of a feedback-based control system was the flyball governor designed by James Watt. This device was used to control the speed of a steam turbine [3].

An evolutionary stride forward in the theory of control systems came from the Scottish mathematician and physicist James Clerk Maxwell in 1868. Using differential equations, Maxwell was able to develop a mathematical model to explain the behavior of Watt's flyball governor. The work of Maxwell demonstrated the significant benefit of using mathematics to explain complex behavior, such as dynamic systems [2]. This point in history reshaped the world of control systems and laid the foundation for today's work which relies heavily on the application of mathematics.

Two distinct paths emerged in the mathematical study of control systems. The western world was primarily focused on frequency-domain analysis whereas the eastern world concentrated on time-domain analysis [3]. Both approaches provided significant rise to the theoretical study of control systems. Progress continued to be made in the advancement of control theory with the next major leaps taking place during World War II. The onset of the war brought forth a demand for new technologies such as gun-positioning systems and radar antenna control systems. Prior to this point, much of the design of control systems had been based on guess work; however, the demand for advanced systems contributed to a need for more precise methods which gave birth to the field of control engineering [3]. Today, we continue to make great strides in the development of control application and theory.

1.2 Survey of Control Methods

In general, we typically classify control systems into one of two main categories: classical control and modern control. A third classification called neoclassical was proposed in [4]. This section discusses these three types of control methods.

1.2.1 Classical Controls

The classical approach to controller design relies on the use of gains and the addition of poles and zeros to control the response of a closed-loop system. A common, and widely used, classical controller is the Proportional-Integral-Derivative (PID) controller. As implied by the name, the PID controller consists of a proportional component, an integral component, and a derivative component. The proportional component is a gain that is proportional to the error of the response (error is defined here as the difference between the desired and actual output of the system). The integral and derivative components also consist of gains which are used to add poles and zeros to the system. The proportional, integral, and derivative gains, along with the feedback loop, alter the locations of the poles and zeros in the closed-loop transfer function to achieve a desired response. The PID controller is discussed further in Chapter 3.

While the PID controller is an effective means of controlling a system, the design process can be difficult. A source of this difficulty arises when deciding on where the closed-loop poles and zeros should be located. A controller's effectiveness is typically evaluated on several performance characteristics including overshoot, peak time, settling time, rise time, and steady-state error. These performance characteristics (which are explored in Chapter 2) are determined by the location of the closed-loop poles and zeros. While mathematical methods exist for second order systems to determine closed-loop pole/zero locations that will ensure specific performance characteristics are met; no such methods exist for higher ordered systems. Often, in higher ordered systems, pole and zero locations are initially estimated and then adjustments are made until a desirable response is achieved. Making these adjustments can also be difficult because moving a single pole, or zero, can improve a particular performance characteristic while having a negative impact on another. As a result, the design process for higher ordered systems often relies on the intuition of an experienced designer and iterative trial-and-error.

1.2.2 Modern Controls

Similar to classical methods, the modern approach to controller design uses gains to control a system's output. The difference between the classical and modern approach is where the gains are located. Modern control places the gains in the feedback loop of the control system. The gains are applied to the individual states of the system and have the effect of moving system poles while leaving any zeros unaffected.

A caveat to this technique is that in order to move the poles of the system, the system must be controllable. A controllable system is one that is capable of driving any of the system's states to any desired value in a finite time through some input [5]. If the system is controllable, then we can use state feedback gains to move the poles of the system to any arbitrary location (provided complex poles are assigned in conjugate pairs). State variable feedback is discussed in further detail in Chapter2.

Another consideration comes from our ability to measure the states of the system. Since we want to use the states of the system to alter the poles, it follows that we must be able to measure the states. However, it is often the case that some of the states of the system are not able to be measured, or inferred. This can happen because sensors needed to measure a particular state might not exist or may be too expensive [5]. If a state cannot be measured, or inferred, a second system called an observer can be designed. The observer outputs an estimate of the system's states. In order to implement an observer the system must be observable. A system is said to be observable if it is possible to determine an unknown initial state in a finite time from knowledge of the input and output history [5]. If properly designed, it is guaranteed that the observers estimate will converge to the actual state values. Observers are discussed further discussed in Chapter 2.

If the system is controllable, then it is fairly simple to determine the gains necessary to place the system's poles in any location we desire. However, we must first know what locations will result in a desirable response. Similar to the classical case, the task is to find locations for the poles such that all performance characteristics are within an acceptable range for the system. Mathematical methods exist for second order systems to determine closed-loop pole locations that will ensure specific performance characteristics are met; however, no such methods exist for higher ordered systems. As was the case with the classical approach, the design process for higher ordered systems often ends up being trial-and-error based. Additionally, there is another obstacle when using the modern approach to controller design; we have no way to change the zeros of the closed-loop system.

1.2.3 Neoclassical Controls

The neoclassical approach proposed in [4] combines classical and modern approaches to controller design. The goal of the neoclassical method is to match the closed-loop transfer function of a system to a transfer function that has already been optimized to a specific performance index. State variable feedback is used to move the closed-loop poles of the system to match the location of the optimized transfer function's poles. A pre-compensator is then used to match the gain of the system to the gain of the optimized transfer function and remove any zeros. The neoclassical technique, which is discussed in detail in Chapter 2, can simplify the design process because the form of the closed-loop transfer function is known in advance. The difficulty discussed earlier with regard to determining appropriate locations for the closed-loop poles and zeros is eliminated because these locations are pre-determined. It was shown in [4] through several simulations that the neoclassical design methodology can be successful in continuous time.

A later work took the neoclassical design technique a step further and explored the effectiveness of the neoclassical approach on discrete systems [6]. The methods used to design discrete controllers under the neoclassical framework are essentially the same as the methods used to design continuous time neoclassical controllers. The only significant difference is the discrete time system must first be discretized using one of several discretization techniques. It was shown through a number of simulations that the neoclassical approach is indeed, in theory, capable of designing a discrete time controller efficiently. The work done in [6] also included the design of a discrete time observer. Like the controller, the observer was also designed based on form of a transfer function optimized for a specific performance index. Several simulations were preformed to show that the theoretical observer design was indeed successful.

The neoclassical controller showed promising simulation results in both continuous time and discrete time systems. However, promising simulation results do not necessarily guarantee success on a physical system. To this point, a controller designed under the neoclassical framework has never been used to control a physical system. Additionally, the neoclassical method has never been compared to other controller design methodologies or implementations.

1.3 Survey of Benchmark Systems

In the process of designing a controller, it is desirable to evaluate the controller's effectiveness on a physical system. Generally speaking, a benchmark is a standard to which alternate designs can be compared. Benchmark systems can be used to validate experimental controller design. The experimental controller will be designed to elicit a particular response on a benchmark system. The response of the system can then be used to evaluate how well the controller performed. The experimental controller can then be compared against other controller types deployed on the same benchmark system.

There are several systems that are considered to be benchmark systems. The mass-spring-damper is one such system. In this system, a mass is attached to a spring and a damper. A force is applied to the mass to cause translational motion. A reference will typically be defined as an input to the system. This input is used to define the desired displacement of the mass. The controller then outputs the force necessary to achieve the desired displacement of the mass. For small displacements, this system is considered to be linear.

Another benchmark system is the inverted pendulum. There are several variations of this system all of which are non-linear. The most common variation is one that places an inverted pendulum bob on a cart. The pendulum is allowed a single dimension of freedom in which it can rotate. A controller is designed to allow the cart to move in such a way that the pendulum remains inverted. A variation of the inverted pendulum called the Furuta pendulum uses rotational motion, instead of translational motion, to keep the pendulum bob inverted.

Magnetic levitation is a non-linear benchmark system that uses an electromagnet to suspend a ball at some distance below the magnet's surface. The controller receives a reference as an input which is used to define the desired distance between the magnetic object and the magnet's surface. The controller will then adjust the current running through the coil that varies the magnetic field allowing of the magnet to achieve, and maintain, this distance.

Two additional linear benchmark systems include the motor position controller, which is used to control the angular position of a motor's shaft, and the motor speed controller, which controls the angular velocity of a motor's shaft.

1.4 Survey of Hardware

Before designing and implementing a controller on a complex system, it is advantageous to first validate design methodologies on less complex hardware. To this end, the development of a testbed for the purposes of validating experimental controller designs becomes highly attractive. Several testbed options are available for purchase through the company Acumen Labware.

Acumen Labware markets products that are primarily intended for use as instructional aids; however, these products could also be used for controller validation. Most of the benchmark systems discussed earlier are available for purchase through Acumen Labware. A potential drawback to the products offered by Acumen Labware is that most, if not all, of their products do not include the means to construct an automatic controller. For example, a magnetic levitation system is available; however, the only means of controlling the current is through a manually operated potentiometer. Terminals are available that would allow for the connection of an external controller; however, the materials needed to construct such a controller are not included. Data acquisition may also be a potential drawback to Acumen Labware's Magnetic Levitation system. While digital displays are included to provide real-time current and position measurements, there is no internal means provided to record the data. Acumen Labware is not unique in the sense that there are other companies that market similar products and have similar limitations.

Another option would be to construct a testbed using a microcontroller and appropriate hardware. Microcontrollers such as the PSoC, Arduino, or PIC are highly popular due to their low-cost and flexibility. The large variety of microcontrollers available make control of any of the benchmark systems discussed earlier possible. However, there is a drawback to constructing a testbed using a microcontroller. Microcontrollers typically require some working knowledge of a coding language, or familiarity with additional software that is capable of generating code. If the testbed is to be used by multiple persons, then all must be familiar with whatever coding language is required to program the microcontroller or be able to generate the code by some other means. What is currently missing is a testbed that is easy to use and inclusive (contains all components necessary to implement and evaluate a controller).

National instruments and Quanser provide another solution. The National Instruments ELVIS III board is a modular engineering educational laboratory device developed specifically for academia [9]. The ELVIS III board comes equipped with several common laboratory test equipment including oscilloscope, function generator, power supply, and digital multi-meter. It is possible to create controllers, using software developed by National Instruments, and deploy them to the ELVIS III board. Quanser makes several systems that can be easily attached to the ELVIS III as a top-mounted board which allow the user to control of electrical and mechanical systems. Some of the systems Quanser offers are the benchmark systems mentioned in the previous section. These systems include the motor position/speed control, magnetic levitation, inverted pendulum (both traditional and Furuta variations), and various mechatronic systems. The combination of the ELVIS III and Quanser top board require a very small footprint and boast ease-of-use.

1.5 Scope of Work

This thesis serves to develop an efficient method to experimentally validate controllers designed under the neoclassical framework. The proposed testbed is used to validate three controllers that are designed using a neoclassical control technique. An assessment of the testbed's usefulness in validating the neoclassical controllers is made and an evaluation of the neoclassical controllers' suitability as physical controllers is presented.

This thesis is organized as follows; chapter two discusses the ELVIS III and Quanser Controls Board and outlines the neoclassical design methodology. Chapter three consists of the implementation of the neoclassical controller on the testbed. Additionally, a PID controller will be designed and implemented on the testbed for comparative purposes. Chapter four discusses the advantages and disadvantages of the testbed and the effectiveness of a neoclassical controller on a physical system. Chapter five summarizes the thesis and discusses future work.

Chapter 2: Testbed Hardware and Neoclassical Design Methodology

One of the objectives of this thesis is to develop a testbed that can be used as an efficient means to experimentally validate new controller designs. The National Instruments ELVIS III board coupled with the Quanser Controls Board is used to accomplish this. The testbed is used to evaluate three experimental controllers which were designed using the neoclassical framework. This chapter begins by discussing the National Instruments ELVIS III and Quanser Controls boards. Next, the neoclassical approach to controller design is discussed.

2.1 ELVIS III

The Educational Laboratory Virtual Instruments Suite III (ELVIS III) is an educational tool developed by National Instruments. It is intended to be used as a stand-alone laboratory which allows students to learn various concepts such as analog/digital/power electronics, digital communications, measurements, instrumentation, and controls through a hands-on approach [9]. To achieve this, ELVIS III is equipped with a variety of commonly used measurement and test equipment. The measurement and test equipment are used together with onboard digital and analog I/O, an embedded microcontroller, and an FPGA to provide students with a powerful laboratory experience in a portable package. ELVIS III is designed for ease-of-use that allows students to begin learning quickly without having to master additional skills, such as learning an unfamiliar programming language, to operate the device. Additionally, National Instruments has prepared comprehensive lab manuals that are freely distributed. There are many features available that make ELVIS III appealing to both student and instructor. This section describes some of the features and capabilities of the ELVIS III.

The ELVIS III, shown in Fig. 2.1, has various measurement devices located on the right-hand side of the board. The measurement devices include a Bode plotter, function generator, oscilloscope, I-V analyzer, logic analyzer, pattern generator, variable power supply, and digital multimeter. To operate the measurement devices, the user must first connect the ELVIS III to a host computer using either a USB or an Ethernet connection. Once connected, the user can interface with the measurement devices in either offline or online mode. Offline access requires the user to download support files to the host computer prior to first use that are used to interface with the measurement devices directly. The benefit to offline access is that no internet connection is needed. Alternatively, the user has the option to access the measurement devices through a website operated by National Instruments [10]. Online access does not require additional software to be downloaded to the host computer.

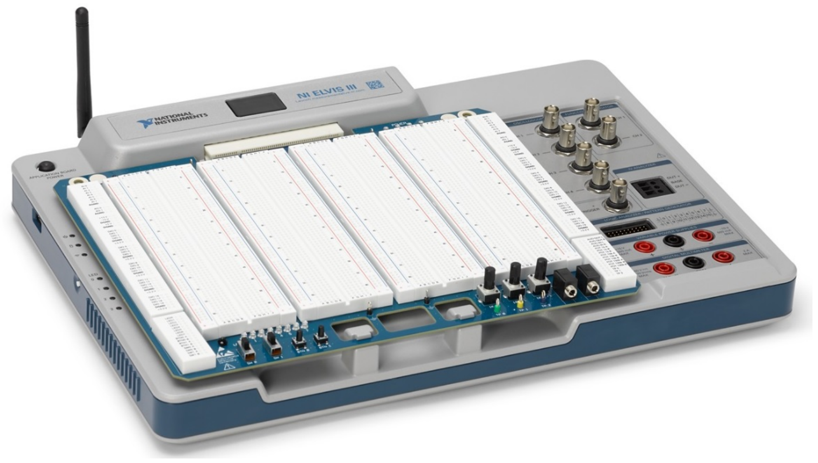


Figure 2.1: ELVIS III [11]

In addition to measurement equipment, ELVIS III comes equipped with a prototyping board. Electrical components such as slide switches, push buttons, potentiometers, and LED lights are available on the lower portion of the board. These components can be used for construction of various experimental circuits. All of ELVIS III's included electrical components can be wired using ports located on the prototyping board. The ports located on the outside columns of the board provide the user with a fixed DC power supply ($+5$ and ± 15 volt) as well as ports to gain access to analog and digital I/O. The ELVIS III can be used with a computer connected to the internet to test basic experimental circuits without the need for additional software; however, additional features can also be accessed through National Instrument's LabVIEW software that allow for far more advanced experimentation.

LabVIEW software provides the user the ability to program the embedded processor and FPGA on the ELVIS III. Typical software packages used to program embedded processors or FPGAs require the user to know a programming language such as C or VHDL. Because there are so many programming languages in use, a learning curve may be necessary for someone who is not familiar with a particular language. In contrast, LabVIEW uses a graphical programming environment to create programs which are referred to as virtual instruments, or VIs. The benefit is that the user need only possess a basic knowledge of what they are working with in order to create a VI. Additionally, LabVIEW provides a graphical interface that allows the user to interact with a running VI. An example of a simple VI that converts degrees to radians is shown in Figs. 2.2 and 2.3. The front panel (Fig. 2.2) is what allows the user to interact with the VI. In this example, the user is able to select an angle in degrees by clicking the slide bar and dragging it to the desired degree. The conversion is computed and then returned to the numerical output. The block diagram (Fig. 2.3) shows how the conversion takes place. The border in the block diagram represents a while loop which will continually execute everything contained within it until the ”Stop VI” button on the front panel is clicked. The user can continually move the slide to obtain a new output until the ”Stop VI” button has been clicked. This is a simple example that demonstrates the graphical approach that is taken in LabVIEW programming. It is possible to create far more sophisticated VIs using LabVIEW. Because LabVIEW is fairly intuitive, it is a good choice to use with the testbed that is developed in this work.

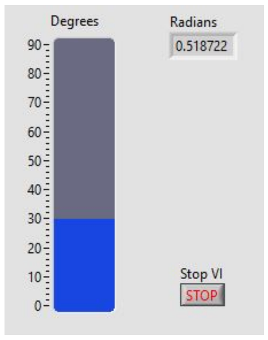


Figure 2.2: Front panel of degree to radian conversion VI

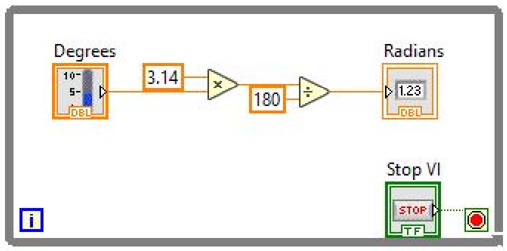


Figure 2.3: Block diagram of degree to radian conversion VI

The capabilities of ELVIS III are greatly increased because of the ability to remove the prototyping board and install specific applications boards in its place. There are a wide variety of application boards that are available that cover a wide range of teaching topics [9]. Some application boards that are currently available are the Communications Board made by Emona, the Power Electronics Board from Texas Instruments, and the Controls Board made by Quanser. Like the ELVIS III, comprehensive lab manuals have been specifically designed for all application boards.

While the ELVIS III is designed primarily for teaching, another use is found in research. ELVIS III gives the user the ability to develop applications that are not defined in the laboratory coursework. Because of this freedom and the wide variety of applications boards available, the ELVIS III is useful in the field of research because experimental systems can be designed and validated using the ELVIS III and an application board.

2.2 Quanser Controls Board

The Quanser Controls Board, shown in Fig. 2.4, is one of the third-party application boards mentioned in the previous section. The purpose of the Controls Board is to promote the study of fundamental and advanced control systems [12]. To this end, Quanser has developed laboratory curriculum which is freely distributed. The labs begin with basic concepts, such plant modeling, and gradually progress to more advanced topics, such as optimal control of an inverted pendulum. They are designed in such a way that someone need only possess a basic knowledge of electrical circuits to be able to use the labs. This makes the labs approachable to a population larger than just electrical engineers.

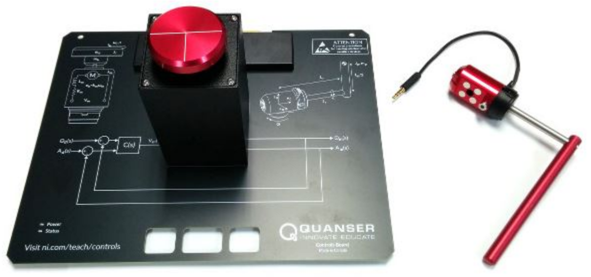


Figure 2.4: Quanser Controls Board [12]

The Controls Board consists of a highly linear DC motor which is mounted at the top of a rectangular enclosure. The housing of the motor is contained within the enclosure so that only the shaft is accessible. Connected to the shaft is a load hub which is used to magnetically attach an inertial disc to the motor. An optional Furuta pendulum accessory is available which can be attached to the load hub by first removing the inertial disc. A high resolution encoder (2,048 counts per revolution) is located in the rectangular housing and is used to determine the radial position of the motor shaft. A second high resolution encoder (also 2,048 counts per revolution) is located in the pendulum assembly and is used to measure the angular position of the pendulum bob. Motor control systems are designed using LabVIEW software and deployed to the ELVIS III. A VI created by Quanser is used to route the control signals generated by the ELVIS III to the input terminals of the motor. A second Quanser VI is used to convert the signal from the encoders to a radial position which is used for feedback in the control system.

2.3 Neoclassical Control Design

The main benefit to the neoclassical control design technique, proposed in [4] and [6], is that the design process eliminates iterative trial-and-error practices common to the classical and modern approaches to controller design for higher order systems. Classical methods depend on controller gains that are typically placed in series with the plant in a closed-loop system to alter the location of the system's poles and zeros. As discussed in the previous chapter, it is sometimes difficult to adjust the gain to achieve a desirable response from the system. This is particularly true in higher ordered systems.

The modern approach also uses gains but places them in the feedback loop of the system. These gains allow the poles to be moved to a location that produces a desirable response; however, they do nothing to alter the zeros. Additionally, for higher ordered systems, it is usually not a simple task to determine where the poles should be placed to achieve the desired response. In both cases (classical and modern), it is common practice for the designer to use experience and iterative trial-and-error guesswork to adjust (or tune) the controller gains to achieve the desired result.

The neoclassical approach to controller design incorporates methods from modern and classical methodologies. In short, the neoclassical technique relies on the ability to alter the transfer function of a system to match an optimized transfer function. This section discusses the details of the neoclassical design process. To gain an understanding of the neoclassical technique proposed in [4] and [6], it is necessary to provide some background regarding the individual components which are used. First, we will discuss how a system is evaluated and introduce the notion of optimized transfer functions. Next, modern and classical control methods will be reviewed. The individual components are then combined to illustrate the neoclassical design technique.

2.3.1 Evaluation of System Performance

The performance of a controlled system is commonly judged in terms of its response to a step input. There are two aspects of a step response responses to be considered: the transient response and steady-state response. The transient response is a system's reaction to a change in the system. This change could come from several sources including a change to the input or a physical perturbation to the system. The transient response is temporary and will die out over time for stable systems [13]. The steady-state response refers to the system behavior after the transient response has subsided. Another way to regard the different aspects of a step response is to think of the steady-state response as the final state of the system and the transient response as the path the system takes to get to the steady-state.

Because the desired steady-state response of a system is generally a design requirement, it is easy to determine how well the system meets this performance criterion. The amount a system deviates from its desired steady-state response is known as steady-state error. Steady-state error is typically expressed as a percent which is commonly referred to as percent error.

Evaluation of the transient response is a bit more complex. There are several metrics that are used to judge the transient response of a system including settling time, rise time, peak time, and percent overshoot. Settling time is a measure of the time it takes for a system's output to fall within a defined upper and lower bound of the desired output and remain there indefinitely. The bounds are given in terms of a percentage of the desired output. For example, the two percent settling time of a system whose desired output is 100 units would be defined as the time it takes for that system to consistently maintain an output in the range of 98 and 102 units.

Rise time is a measurement of how quickly the system responds to a deviation from an equilibrium or steady-state. While any interval may be considered, it is common to choose the interval of time in which it takes a system to get from 10% to 90% of its desired output. For example, consider a system that is in steady-state and has an output of zero units. If the system is designed to output 100 units when a step input is applied, then the rise time would be the time it takes that system to go from 10 units to 90 units.

Peak time is a measure of the length of time taken for the response of the system to reach its maximum, or peak, value. For example, consider a system whose response reaches a maximum of 10 units before eventually reaching the steady-state. The peak time would simply be the time at which the maximum value of 10 units is achieved.

Percent overshoot, in contrast to settling time, rise time, and peak time, does not consider a time metric. Instead, percent overshoot refers to the maximum magnitude in which the output of a system deviates from the desired output. For example, if the desired system output is 100 units and the actual output first reaches 110 units before gradually reaching steady-state, then we would say the system has a 10 percent overshoot.

We must be careful in how we interpret the performance criteria we have discussed so far. Not all systems are the same and some applications may value certain criteria over another. For example, a highly precise surgical robotics application may require the response to provide no overshoot whatsoever while the speed with which the system responds doesn't matter. In such an instance, an overshoot of a fraction of a percent might be considered to be extremely large and a rise time measured in terms of minutes may be insignificant. Conversely, another system may regard a percent overshoot of 10 percent to be acceptable and a rise time of 0.01 seconds to be far too slow. While these metrics are important in evaluating the performance of a system, their evaluation depends on the application and are, therefore, somewhat qualitative in nature.

2.3.2 Optimal Transfer Functions

The performance criteria discussed are very helpful for comparison of responses later in this thesis. There is another performance evaluation method that we must now consider. A performance index, or cost function, can be used to assess system performance. A performance index is a mathematical expression that provides a strictly quantitative measurement to system performance. A system is said to be optimized when evaluation of the cost function returns an extreme value (minimum or maximum) [3]. In some instances, it is desirable to maximize the performance index; however, it is far more common to minimize its value.

There are many different performance indices that are commonly used. One such performance index is the Integral of Time multiplied by the Absolute value of the Error, or ITAE. This is the index on which we will focus. The ITAE performance index is shown in (2.1). The ITAE considers the magnitude of the difference between the actual output and the desired output of the system, $e(t)$. The difference is then weighted with respect to the time at which it occurred. The integral sums up all the weighted differences. What is important about the ITAE criterion is that it places a larger penalty on errors that occur later in time than the errors that occur before the transient response has had time to subside.

$$ITAE = \lim_{A \to \infty} \int_0^A t|e(t)|dt \tag{2.1}$$

The ITAE performance index has been studied extensively. Graham and Lathrop published a paper in the 1953 entitled "The Synthesis of 'Optimum' Transient Response: Criteria and Standard Forms," in which they derive general expressions that optimize the ITAE performance index [14]. The general s-domain transfer function derived in [14] is shown in (2.2) where ω_n is the natural frequency, m is the order of the system, and b_i are the coefficients of the powers of s. The equations shown in (2.3) were provided in [14]. These equations are the denominators of (2.2) for systems of orders 1-4 and are used to determine the values for b_i.

$$T_{ITAE}(s) = \frac{\omega_n^m}{s^m + b_{m-1}\omega_n s^{m-1} + \cdots + b_1 \omega_n^{m-1} s + \omega_n^m} \tag{2.2}$$

$$\begin{gathered} s + \omega_n \\ s^2 + 1.4\omega_n s + \omega_n^2 \\ s^3 + 1.75\omega_n s^2 + 2.15\omega_n^2 s + \omega_n^3 \\ s^4 + 2.1\omega_n s^3 + 3.4\omega_n^2 s^2 + 2.7\omega_n^3 s + \omega_n^4 \end{gathered} \tag{2.3}$$

Examining equations (2.2) and (2.3), we note that the only unknown is the natural frequency, ω_n. The natural frequency can be determined by using the relationship shown in (2.4) where $T_{s,\ norm}$ is the 2% settling time of the appropriately ordered normalized characteristic ITAE transfer function, and $T_{s,\ desired}$ is the desired 2% settling time of the system to be controlled. The normalized settling time, $T_{s,\ norm}$, is found by setting the natural frequency of the characteristic ITAE transfer function to unity. The only variable that is unknown is the desired settling time of the system that is to be controlled. This choice is made by the designer to satisfy the need of the system.

$$\omega_n = \frac{T_{s,\ norm}}{T_{s,\ desired}} \tag{2.4}$$

The benefit to using the ITAE performance index is that there is only a single parameter that the designer needs to consider. Once the designer has chosen an appropriate settling time for the system, the optimized transfer function can easily be determined. We can then use state variable feedback and pre-compensation to match the closed-loop transfer function of the system we wish to control to the optimized transfer function. There is no need to consider the transient response characteristics of percent overshoot, rise time and steady state error because they are determined. It is important to note that the ITAE optimized transfer function of (2.2) allows the designer to have explicit control of the settling time. Other performance characteristics such as percent overshoot, are pre-determined and are not able to be altered by the designer. If percent overshoot is a main design concern, then the ITAE optimized transfer optimized transfer function may not yield an acceptable response.

2.3.3 State Variable Feedback

The contribution that the modern approach lends to the neoclassical method comes from placing gains in the feedback loop which results in what is referred to as state variable feedback. The poles of the closed-loop system can be moved by using state variable feedback. As mentioned earlier, in order to move the poles of the system to any arbitrary location, the system must be controllable. A system is controllable if there exists some input that is capable of driving the system states to any value in a finite time. To determine if a system is controllable, consider the general single-input single-output open-loop system shown in Fig. 2.5 whose equations are defined in (2.5). The controllability matrix shown in (2.6) can be formed using (2.5) and evaluated to determine if the system is controllable. If the controllability matrix is of full rank, then the system is controllable.

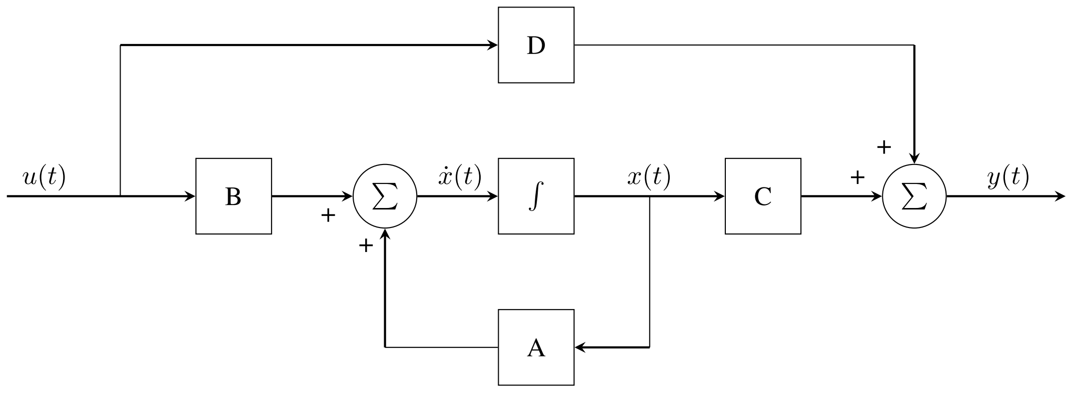


Figure 2.5: State space representation of a system

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t)\\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}u(t)\end{aligned} \tag{2.5}$$

$$\mathscr{C} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^{n-1}\boldsymbol{B} \end{bmatrix} \tag{2.6}$$

If the system is controllable, then we can use state variable feedback to move the poles of the closed-loop system to any location we choose (provided complex poles are assigned in conjugate pairs). To begin, we first modify the block diagram shown in Fig. 2.5 to include the state variable feedback. The resulting modification is shown in Fig. 2.6. If we have access to all the states, then we can allow the input to take the form shown in (2.7) where $\boldsymbol{K}$ is the state variable feedback gain matrix and $r(t)$ is a reference input.

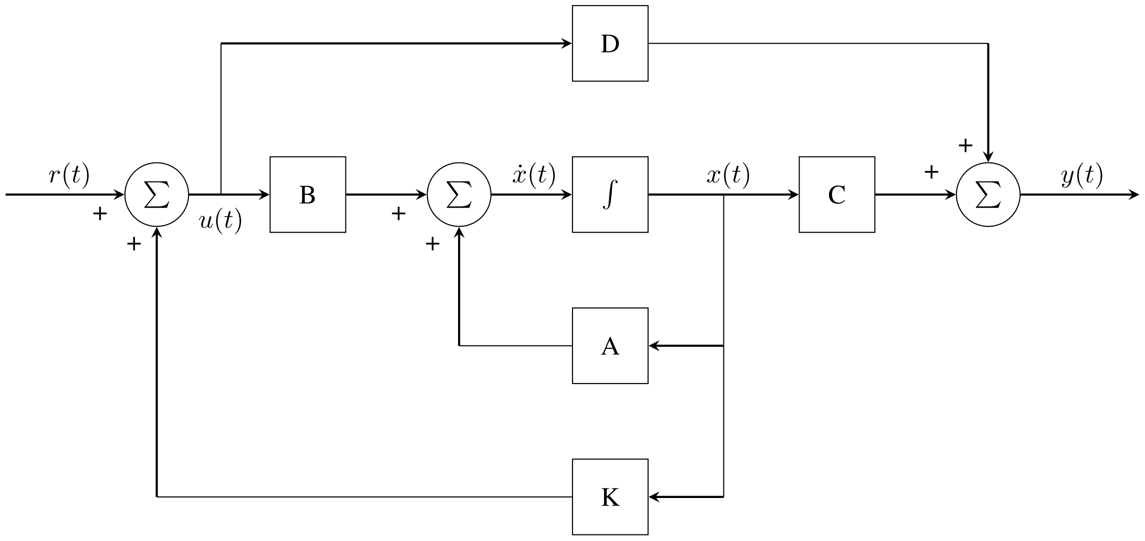


Figure 2.6: Block diagram of a control system with state variable feedback

$$u(t) = \boldsymbol{Kx}(t) + r(t) \tag{2.7}$$

Substituting (2.7) into the state equation of (2.5) yields;

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{Ax}(t) + \boldsymbol{B}u(t) \\ &= \boldsymbol{Ax}(t) + \boldsymbol{BKx}(t) + \boldsymbol{B}r(t) \\ &= (\boldsymbol{A} + \boldsymbol{BK})\boldsymbol{x}(t) + \boldsymbol{B}r(t) \end{aligned} \tag{2.8}$$

Since the poles of the closed-loop system are equivalent to the eigenvalues of the $A + BK$ matrix, we can now place the closed-loop poles at a location of our choosing by carefully selecting the K matrix. Using this technique, we can move the closed-loop system poles to match the pole locations of an optimized transfer function such as the ITAE.

2.3.4 State Estimators

The application of state variable feedback to a system requires that we be able to measure all of the states either directly or indirectly. In practice, there may be cases where we are unable to measure the states. When the states cannot be determined, it is possible to estimate them by introducing a second system called an observer.

In order to design a useful observer, the system needs to be observable. An observable system will ensure that the error in the state estimation will converge to zero. In other words, the state estimation will converge to the actual value of the state. If the system is not observable, then we cannot guarantee that the state estimate will converge to the actual value.

A system is said to be observable if we are able to determine what the initial state of the system is from the history of the input and output [3]. The observability matrix shown in (2.9) is used to determine if a system is observable. If the observability matrix is of full rank, then the system is observable.

$$\mathscr{O} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \vdots \\ \boldsymbol{CA}^{n-1} \end{bmatrix} \tag{2.9}$$

A block diagram for an observer is shown in Fig. 2.7 where the estimated state is denoted using the circumflex (i.e. $\hat{x}$). The block labeled ”plant” is a consolidation of the system shown in Fig. 2.5 where we have assumed D to be the zero matrix for simplicity. The block L represents the observer gain. The equation used to describe the observer is shown in (2.10) and simplified to a common form in (2.11). Examination of (2.10), the last term in particular, reveals the way in which the observer operates. The observer gain matrix, L, essentially places a weight on the difference between the actual output of the system, $y(t)$, and the estimation of the output, $\hat{y}(t) = C\hat{x}(t)$. If the outputs are the same, then no correction is needed and the term goes to zero; however, if the difference is non-zero then a correction will be added to the state estimation.

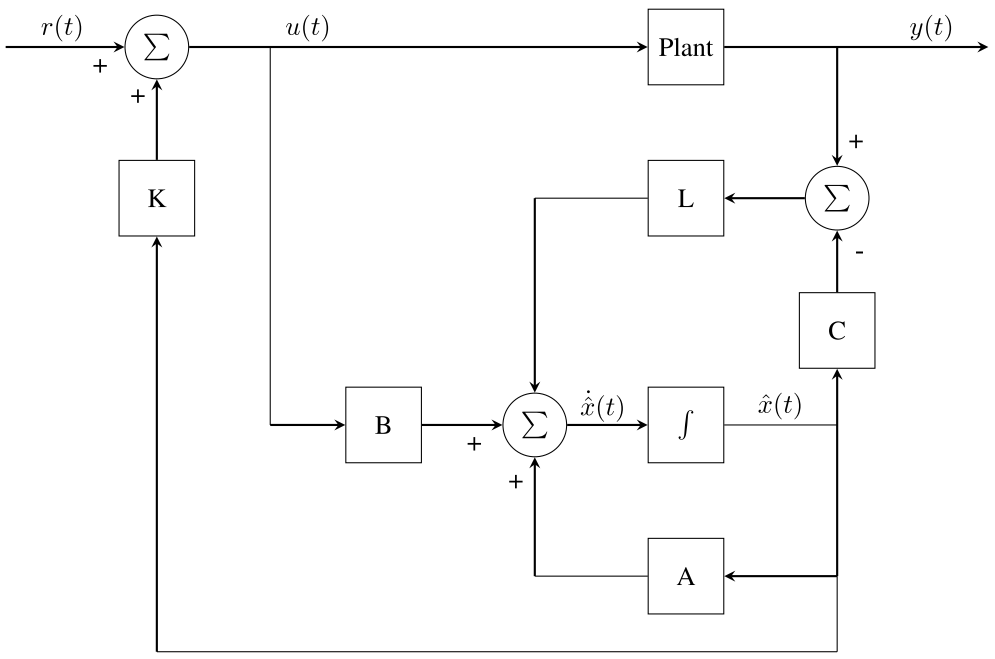


Figure 2.7: Block diagram of an observer with state variable feedback

$$\dot{\hat{\boldsymbol{x}}}(t) = \boldsymbol{A}\hat{\boldsymbol{x}}(t) + \boldsymbol{B}u(t) + \boldsymbol{L}\left(y(t) - \boldsymbol{C}\hat{\boldsymbol{x}}(t)\right) \tag{2.10}$$

$$= (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}}(t) + \boldsymbol{B}u(t) + \boldsymbol{L}y(t) \tag{2.11}$$

When using an observer to estimate the states of a system, it is important to be sure that the estimates quickly converge to the true state values. It is the observer gain, L, which ensures this happens. To show this, we define the error, $e(t)$, as the difference between the actual, $\boldsymbol{x}(t)$, and estimated states, $\hat{\boldsymbol{x}}(t)$.

$$e(t) = \boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)$$

Taking the derivative of both sides and making substitutions from (2.5), where we have set $\boldsymbol{D}$ to be the zero matrix for convenience, and (2.11) gives us:

$$\begin{aligned}
\dot{e}(t) &= \dot{x}(t) - \dot{\hat{x}}(t) \\
&= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) - [(\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}}(t) + \boldsymbol{B}u(t) + \boldsymbol{L}y(t)] \\
&= (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\boldsymbol{x}(t) - (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}}(t) \\
&= (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})e(t)
\end{aligned} \tag{2.12}$$

Equation (2.12) shows that proper selection of the eigenvalues of the $(\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})$ matrix determines how quickly the error (difference between actual and estimated states) converges to zero. Because we want the error to reach zero quickly it is necessary to select the observer gain such that the response of the observer is faster than the response of the system. It is typical to design the observer to be five to ten times faster.

2.3.5 Pre-compensation

Thus far, we have only been concerned with the denominator of the controlled system's closed-loop transfer function. In particular, we have focused on methods to move the closed-loop poles of a system to match the pole locations an optimized transfer function. To accomplish this, we used state variable feedback which is a technique associated with the modern approach to controller design. We now turn our focus to classical control techniques which will allow us to match the numerator of a controlled system's transfer function to an optimized transfer function.

The classical approach relies the use of any combination of gains, poles, and zeroes to alter the closed-loop transfer function of a system. By including some, or all, of these elements in the controller, the numerator of the transfer function can be modified. A general expression for a transfer function of a single-input single-output closed-loop system with state variable feedback can be derived using the Laplace transforms of the output equation in (2.5) and the state equation in (2.8).

$$\begin{aligned}
\boldsymbol{X}(s) &= (s\boldsymbol{I} - (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}))^{-1}\boldsymbol{B}R(s) \\
Y(s) &= (\boldsymbol{C} + \boldsymbol{D}\boldsymbol{K})\,\boldsymbol{X}(s) + \boldsymbol{D}R(s) \\
&= (\boldsymbol{C} + \boldsymbol{D}\boldsymbol{K})\,(s\boldsymbol{I} - (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}))^{-1}\boldsymbol{B}R(s) + \boldsymbol{D}R(s) \\
&= \big((\boldsymbol{C} + \boldsymbol{D}\boldsymbol{K})\,(s\boldsymbol{I} - (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}))^{-1}\boldsymbol{B} + \boldsymbol{D}\big)\,R(s) \\
\frac{Y(s)}{R(s)} &= (\boldsymbol{C} + \boldsymbol{D}\boldsymbol{K})\,(s\boldsymbol{I} - (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}))^{-1}\boldsymbol{B} + \boldsymbol{D} \qquad (2.13)
\end{aligned}$$

In order to match the numerator of the controlled system's transfer function to the numerator of an optimized transfer function, we first cancel the original numerator by setting the pre-compensator denominator equal to the numerator that results from (2.13). Next, we set the numerator of the pre-compensator equal to the numerator of an optimized transfer function. By placing a pre-compensator of this form in series with the plant, we are able to match the numerator of an optimized transfer function.

There are additional considerations to make when designing the pre-compensator. First, it may be necessary to include an integrator in order to eliminate steady state error. The addition of an integrator will add an open-loop pole at the origin to the system and raise the type number. The type number of a system provides an indication as to the amount of steady state error a system will have to a particular input. To

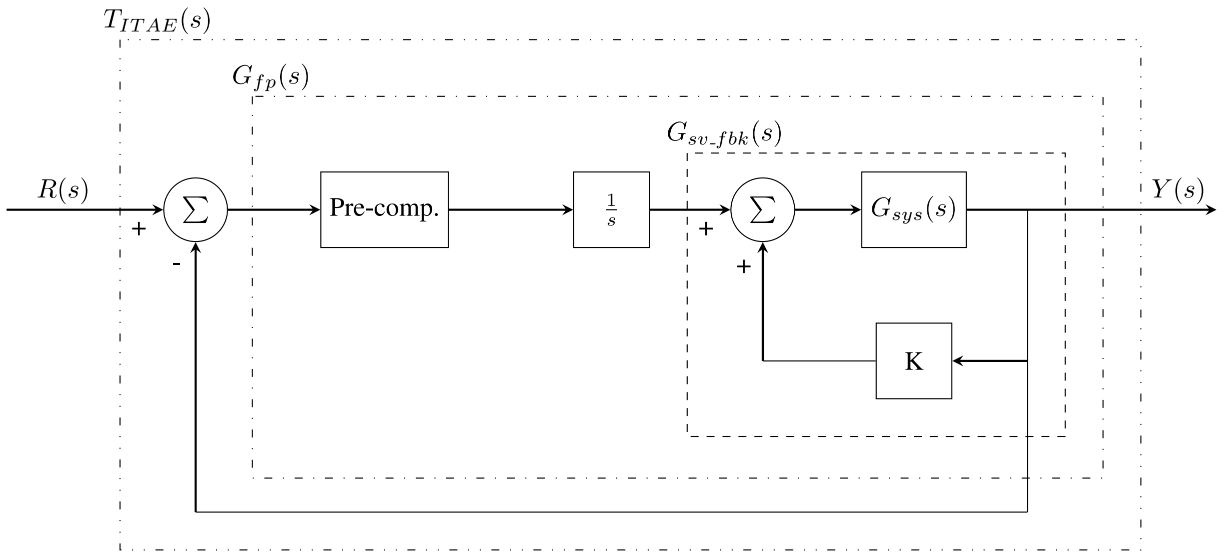


Figure 2.8: Block diagram of a neoclassically controlled System

achieve zero steady state error to a step input a system must be of type one or greater. In this thesis, the plants that are considered are all of type 0. Additionally, it is required that the closed-loop responses have no steady state error to an applied step input. Therefore, an integrator will be included in the controllers considered in this thesis.

Another consideration we need to be aware of is with respect to the location(s) of the zero(s) of the plant. In theory, we are able to mathematically cancel any zero through the addition of a pole; however, in practice it is nearly impossible to exactly cancel a zero. Therefore, attempting to cancel a zero that resides in the Right-Half Plane (RHP) in the s-domain (for continuous time systems) may lead to instability through the introduction of an unstable pole. Similarly, attempting to cancel a zero that resides outside the unit circle for discrete time systems may lead to instability due to the addition of an unstable pole. This implies that, if an uncontrolled plant has an zero that requires cancellation by the addition of an unstable pole, then the neoclassical approach to controller design may fail to yield a stable closed-loop system.

2.3.6 Neoclassical Technique

Having discussed optimal transfer functions, state variable feedback and pre-compensation; we can now put these elements together and consider the neoclassical design process proposed in [4] and [6]. The general continuous time neoclassical block diagram is shown in Fig. 2.8 where the block labeled $G_{sys}(s)$ is the transfer function of the analog process to be controlled, $R(s)$ is the system input, and $Y(s)$ is the system output. It is noted that Fig. 2.8 can be applied to discrete systems by making a conversion from the s-domain to the z-domain.

The portion of the block diagram enclosed by the box labeled T_{ITAE} is our primary focus. This portion is the starting point of the design process. It represent the closed-loop transfer function of the controlled system. When the design is complete, it will be equivalent to the optimized ITAE optimized transfer function discussed previously. Because we know the form of T_{ITAE} by simply selecting a desired settling time, we start from this point and work backwards to determine the form the optimal forward path G_{fp} must take. Working backwards from T_{ITAE} an expression for G_{fp} is shown in (2.14). We can then substitute (2.2) into (2.14) to obtain the general expression for G_{fp} shown in (2.15). The goal is now to use state variable feedback and pre-compensation to force our system to match (2.15). Then, when the system is enclosed by unity feedback the transfer function will match the ITAE transfer function.

$$T_{ITAE} = \frac{G_{fp}}{1 + G_{fp}} \quad \Longrightarrow \quad G_{fp} = \frac{T_{ITAE}}{1 - T_{ITAE}} \tag{2.14}$$

$$G_{fp} = \frac{\omega_n^m}{s(s^{m-1} + b_{m-1}\omega_n s^{m-2} + \ldots + b_1 \omega_n^{m-1})} \tag{2.15}$$

Examination of (2.15) provides two main takeaways. First is with respect to the s term factored from the denominator. The prior discussion of the pre-compensator mentioned including an integrator to eliminate steady state error. Conveniently, the factored s term represents the continuous time integrator which is defined as ${}^1\!/_s$. Next, we consider the expression enclosed by parenthesis in the denominator and note that it is of order $m - 1$. This implies that our system is also of order $m - 1$ since the integrator contributes the additional s. Therefore, when we select the characteristic ITAE equation from (2.3), we need to be sure we select one order higher than that of our original system to obtain the appropriate coefficients. In order to proceed using the neoclassical method all that is needed at this point is to apply state variable feedback and pre-compensation. Because all parameters in (2.15) are known, the state variable gain matrix, K, and pre-compensator can be determined analytically. Alternatively, the *place* command in MATLAB can be used to determine the appropriate state variable gain matrix.

The design process in discrete time is nearly identical with the only difference being the need to discretize both the plant and the ITAE optimized transfer function. There are several methods that can be used to discretize a system. One of the more popular methods is known as the zero-order hold, or ZOH. The equations used to discretize a system represented in state space using ZOH are shown in (2.16) where T is the sampling time of the system and the subscript d denotes a discretized matrix. Note that the $\boldsymbol{C}$ and $\boldsymbol{D}$ are unchanged by this discretization method.

$$\boldsymbol{A}_d = e^{\boldsymbol{A}T} \qquad \boldsymbol{B}_d = \int_0^T e^{\boldsymbol{A}\tau} d\tau \qquad \boldsymbol{C}_d = \boldsymbol{C} \qquad \boldsymbol{D}_d = \boldsymbol{D} \tag{2.16}$$

The state and output equations are shown in (2.17).

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{A}_d\boldsymbol{x}[k] + \boldsymbol{B}_d u[k] \\ \boldsymbol{y}[k] &= \boldsymbol{C}_d\boldsymbol{x}[k] + \boldsymbol{D}_d u[k] \end{aligned} \tag{2.17}$$

It is also necessary to discretize the integrator. The conversion of an integrator from s domain to z domain is shown in (2.18). Here, the integrator is discretized using the Euler forward difference method which is discussed in more detail.

$$\frac{1}{s} = \frac{T}{z-1} \tag{2.18}$$

Since it is difficult to obtain a general expression for the discrete time ITAE transfer function, we will derive the expressions for the appropriately ordered system when needed in the implementation section.

There are several other discretization methods available; however, the ZOH method is preferable because at every time step, Tk, where T is the sampling time and k is the time index; the discrete state and output equations shown in (2.17) exactly match the continuous state and output equations shown in (2.5). Unfortunately, there are some limitations with the ability to use the ZOH method for neoclassically controlled systems. In some instances, the discretization can introduce zeros to the system that need to be canceled through the addition of unstable poles. As discussed earlier, since we cannot perfectly cancel zeros, the addition of unstable poles may lead to instability. An alternative is Euler's forward difference method.

Euler's forward difference method relies on the fact that a derivative can be approximated as follows;

$$\dot{\boldsymbol{x}}(t) = \lim_{T\to 0} \frac{\boldsymbol{x}(t+T) - \boldsymbol{x}(t)}{T} \approx \frac{\boldsymbol{x}(t+T) - \boldsymbol{x}(t)}{T} \tag{2.19}$$

Substituting (2.19) into (2.5), we can write discrete time state space equations as shown in (2.20). Note that sampling has been applied to the equations (i.e. $t = kT$ where T is the sampling time and k is the time index).

$$\boldsymbol{x}[(k+1)T] = (\boldsymbol{I} + T\boldsymbol{A})\boldsymbol{x}(kT) + T\boldsymbol{B}u(kT)$$
$$y(kT) = \boldsymbol{C}\boldsymbol{x}(kT) + \boldsymbol{D}u(kT) \tag{2.20}$$

Discretized matrices can now be defined and are shown in 2.21 where T is the sampling time of the system and the subscript d denotes a discretized matrix. As was the case with the zero-order hold method, the $\boldsymbol{C}$ and $\boldsymbol{D}$ matrices are unchanged by this method.

$$\boldsymbol{A}_d = (\boldsymbol{I} + T\boldsymbol{A}) \qquad \boldsymbol{B}_d = T\boldsymbol{B} \qquad \boldsymbol{C}_d = \boldsymbol{C} \qquad \boldsymbol{D}_d = \boldsymbol{D} \tag{2.21}$$

Euler's forward difference method is very easy to implement; however, ease of implementation comes at a cost. In this case the cost is accuracy. Euler's forward difference method can never be exact because it is derived by using an approximation for a derivative. Accuracy can be improved by choosing the sampling time to be small; however, we can not choose the sampling time to be faster than what is computationally possible. It is necessary when using Euler's forward difference method to discretize a system to consider the accuracy needed for the system to operate as desired.

Pole-zero matching is another method that can be use to discretize a continuous time system. The pole-zero matching method relies on the fact that poles from the s-domain can be mapped into the z-domain using (2.22) where p_z is a pole in the z-domain, p_s is a pole in the s-domain, and T is the sampling time of the discrete system.

$$p_z = e^{p_s T} \tag{2.22}$$

Generally, this mapping is used only valid for poles. However, the pole-zero matching method uses (2.22) to map both poles and zeros from the s-domain to the z-domain. Additional zeros are then added to the discretization at $z = -1$ if the s-domain transfer function has more zeros than poles. The general form of a z-domain transfer function discretized using pole-zero matching discretization is shown in (2.23) where K_c is the gain of the continuous time system, α is the DC gain, n and m are the number of zeros and poles in the continuous time transfer function respectively, and a_i and b_j are the i^{th} zero and j^{th} pole of the continuous time transfer function respectively.

$$G(z) = \alpha K_c \frac{(z+1)^{n-m-1} \prod_{i=1}^{n} (z - e^{a_i T})}{\prod_{j=1}^{m} (z - e^{b_j T})} \tag{2.23}$$

The first step to designing a discrete time controller using the neoclassical method is to discretize the system and appropriate ITAE transfer function. The next step is to find $G_{fp}(z)$ from Fig. 2.8. Once $G_{fp}(z)$ has been determined, we then use state variable feedback to adjust the poles and pre-compensation to cancel the zeros, add the appropriate gain and raise the type number of the system. As stated before, the process is nearly identical to the continuous time version. The only difference is the need to first discretize the plant, ITAE optimized transfer function, and integrator.

Chapter 3: Implementation

The purpose of this chapter is to explore how new control system designs are validated using the ELVIS III/Quanser (E/Q) testbed. To accomplish this, three controllers are designed using the neoclassical framework and then deployed to the ELVIS III. In each case, the controllers are designed to control the shaft position of the motor located on the Quanser Controls Board. Controllers are designed in both continuous time and discrete time. A third controller incorporates an observer that will be used to estimate an unknown state. Additionally, a Proportional-Integral-Derivative (PID) controller will be designed and implemented. Because the PID is a popular, well established means of control, it will be compared to the neoclassical method in terms of ease-of-design and performance. In addition to confirming the usefulness of the E/Q testbed, a successful deployment will also validate the neoclassical method as means for controller design.

This chapter begins with the development of a model that is used to describe the dynamics of the motor located on the Quanser Controls Board. Once a model has been developed, a neoclassical controller will be designed in continuous time and deployed to the E/Q testbed. Next, the process is repeated in discrete time. An observer, will then be added to the design. Lastly, a PID controller will be designed and implemented.

3.1 Modeling

Before we can begin designing a controller, we must develop a model for the motor located on the testbed. This model consists of differential equations that explain the dynamics of the motor. Since we are concerned with motion, the differential equations may be referred to as equations of motion. The controllers developed in this thesis are all designed to control the position of a motor. Thus, we must develop a model that can relate an electrical input measured in volts, to an output that is measured in radians. This section discusses the process used to model the testbed's motor. This model will then be used to design three neoclassical controllers and a PID controller.

Consider the schematic of the simple armature controlled motor shown in Fig. 3.1 where $v_m(t)$ is the terminal voltage, R_m is motor resistance, L_m is motor inductance, $i_m(t)$ is motor current, $v_b(t)$ is the back EMF created by the spinning motor, and J_m, J_h and J_d are the moments of inertia for the motor, load hub and inertial disk respectively. Focusing first on the electrical aspect; Kirchhoff's voltage law is used to write equation (3.1).

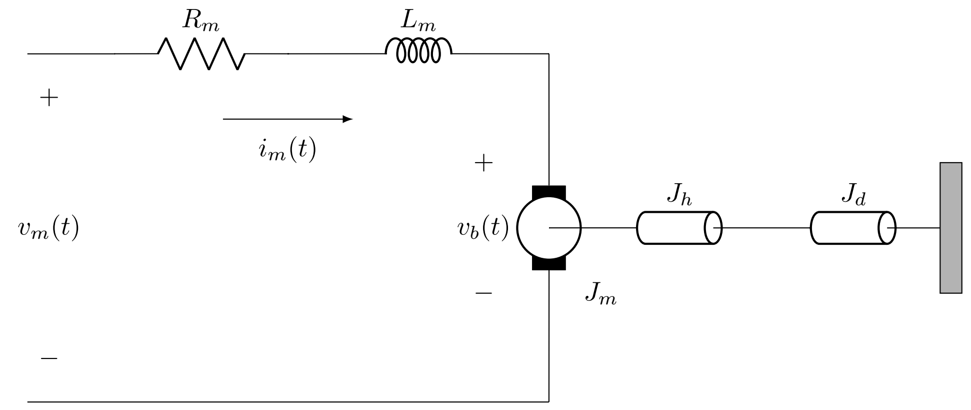


Figure 3.1: Armature controlled motor with inertial load [1]

$$v_m(t) = R_m i_m(t) + L_m \frac{d}{dt} i_m(t) + v_b(t)$$

$$\frac{d}{dt} i_m(t) = \frac{1}{L_m} \left[v_m(t) - R_m i_m(t) - v_b(t) \right]$$

$$\dot{i}_m(t) = \frac{1}{L_m} \left[v_m(t) - R_m i_m(t) - v_b(t) \right] \tag{3.1}$$

The back EMF, $v_b(t)$, is given as the product of mechanical speed of the motor and the motor's back EMF constant. The relationship is shown in (3.2) where k_m is the back EMF constant, and $\omega_m(t)$ is the angular velocity of the motor's shaft as a function of time.

$$v_b(t) = k_m \omega_m(t) \tag{3.2}$$

Substituting (3.2) into (3.1), we arrive at the first equation of motion shown in (3.3). Because we are ultimately concerned with position we replace $\omega_m(t)$ with the first derivative of angular position with respect to time, $\dot{\theta}_m(t)$.

$$\dot{i}_m(t) = \frac{1}{L_m} \left[v_m(t) - R_m i_m(t) - k_m \dot{\theta}_m(t) \right] \tag{3.3}$$

We now focus on the mechanical portion of Fig. 3.1. Using Newton's second law, we can write an expression that describes the torque of the motor. This expression is shown in (3.4), where $\alpha(t)$ is angular acceleration, $\tau(t)$ is torque, and J_{eq} is the total moment of inertia acting on the motor.

$$\tau(t) = J_{eq} \alpha_m(t) \; ; \;\; J_{eq} = J_m + J_h + J_d \tag{3.4}$$

We must now relate the mechanical equation to the electrical equation. Finding such a relation is crucial because it couples the equations which allows the electrical and mechanical equations of motion to influence each other. The relation, shown in (3.5), comes from the fact that motor torque is related to the current of the motor where k_t is the motor torque constant.

$$\tau(t) = k_t i_m(t) \tag{3.5}$$

Substituting (3.5) into (3.4) leads to the second equation of motion which is shown in (3.6). Since we are concerned with position, $\alpha_m(t)$ has been replaced with the second derivative of angular position with respect to time, $\ddot{\theta}_m(t)$.

$$\ddot{\theta}_m(t) = \frac{k_t}{J_{eq}} i_m(t) \tag{3.6}$$

The differential equations shown in (3.3) and (3.6) represent a model of the dynamics of the motor. These equations can be combined in state space form shown in (3.7). The values for the system parameters are shown in Table 3.1

$$\begin{bmatrix} \dot{\theta}_m(t) \\ \ddot{\theta}_m(t) \\ \dot{i}_m(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & \frac{k_t}{J_{eq}} \\ 0 & -\frac{k_m}{L_m} & -\frac{R_m}{L_m} \end{bmatrix} \begin{bmatrix} \theta_m(t) \\ \dot{\theta}_m(t) \\ i_m(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L_m} \end{bmatrix} v_m(t)$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_m(t) \\ \dot{\theta}_m(t) \\ i_m(t) \end{bmatrix} \tag{3.7}$$

Table 3.1: Quanser Controls Board motor parameters

Parameter	Value	Description
R_m	8.4Ω	Motor Terminal Resistance
L_m	$1.16mH$	Motor Inductance
k_t	$0.042^{Nm}/_{A}$	Motor Torque Constant
k_m	$0.042^{V}/_{rad/s}$	Motor Back EMF Constant
J_m	$4.0 * 10^{-6} kgm^2$	Rotor Moment of Inertia
J_d	$1.6 * 10^{-5} kgm^2$	Disc Moment of Inertia
J_h	$0.6 * 10^{-6} kgm^2$	Load Hub of Inertia
m_h	$0.0106kg$	Mass of Load Hub
m_d	$0.053kg$	Mass of Disc
r_h	$0.0111m$	Radius of Load Hub
r_d	$0.0248m$	Radius of Disc

Designing a controller using the neoclassical framework requires that all states in the model can either be directly measured or can be determined indirectly. In the case of the E/Q testbed, a quadrature encoder is used to directly measure angular position and angular velocity can be determined indirectly using this data. However, we have no way to directly measure motor current nor are we able to determine its value using indirect methods. To accommodate the hardware constraints, we will develop a model that does not use current as a state. The simplified model is used for the first two neoclassical controller designs (continuous and discrete time respectively). The full model is used for the third neoclassical design which incorporates an observer. The observer is used to estimate the value of the unknown motor current state.

In order to modify the model so that the current, $i_m(t)$, is no longer a state, we will assume that the voltage drop across the inductor is very small compared to the voltage drop across the resistor. This assumption is reasonable because from Table 3.1 we can see that the inductance of the motor is much less than the resistance. If we ignore the voltage drop across the inductor, then we can re-draw the schematic as shown in Fig. 3.2. Applying Kirchhoff's voltage laws to the circuit gives us equation (3.8) where the back EMF relationship in (3.2) has been used.

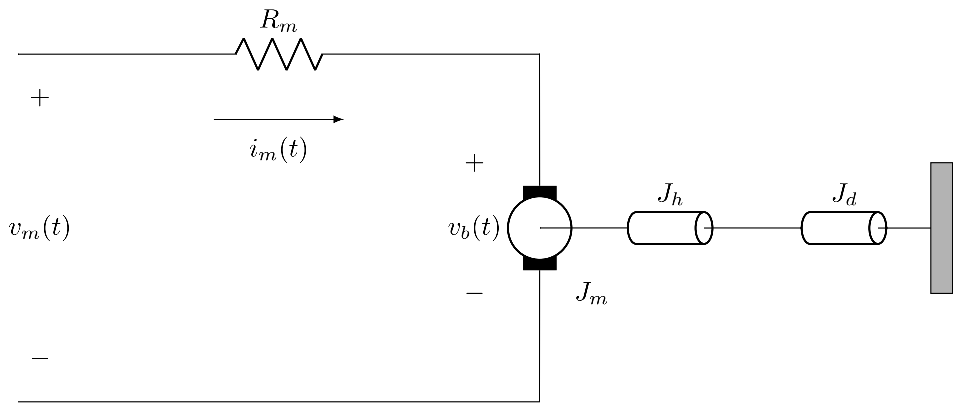


Figure 3.2: Modified armature controlled motor with inertial load [1]

$$i_m(t) = \frac{1}{R_m}\left[v_m(t) - k_m\dot{\theta}_m(t)\right] \tag{3.8}$$

Substituting this result into (3.6) we obtain (3.9). The state space form of the motor model is now shown in (3.10). The current $i_m(t)$ is no longer a state in this representation.

$$\begin{aligned}\ddot{\theta}_m(t) &= \frac{k_t}{J_{eq}} i_m(t) \\ &= \frac{k_t}{J_{eq}R_m}\left[v_m(t) - k_m\dot{\theta}_m(t)\right]\end{aligned} \tag{3.9}$$

$$\begin{bmatrix} \dot{\theta}_m(t) \\ \ddot{\theta}_m(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{k_t k_m}{J_{eq} R_m} \end{bmatrix} \begin{bmatrix} \theta_m(t) \\ \dot{\theta}_m(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_t}{J_{eq} R_m} \end{bmatrix} v_m(t)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta_m(t) \\ \dot{\theta}_m(t) \end{bmatrix} \tag{3.10}$$

3.2 Continuous Time

This section explores the implementation of a continuous time neoclassical controller for the E/Q motor system on the testbed. First, the controller is designed using the neoclassical technique. Next, the controller design is validated through simulation. Finally, the controller is deployed to the testbed.

3.2.1 Controller Design and Simulation

The second order model is used to design the neoclassical controller in continuous time. The goal is to force the transfer function of the closed-loop system to match an ITAE optimized transfer function. Recall from equation (2.15) that in order to successfully match the transfer functions, we need to select the ITAE characteristic function that is an order higher than the system itself. Therefore, we select the third order characteristic function from the list shown in (2.3).

The natural frequency, ω_n, is found using the relation shown in (2.4). Because the desired settling time will be chosen as a specific value based on the application's requirements, the only unknown is the normalized settling time, $T_{s,\ norm}$. The normalized settling time of the third order ITAE optimized transfer function is found by setting the natural frequency to unity and then determining the settling time to an applied unit step input. The plot shown in Fig. 3.3 indicates that the normalized settling time is $T_{s,\ norm} = 7.54s$. For simplicity, we set the desired settling time to be $T_{s,\ desired} = 1s$. We now calculate the natural frequency in (3.11).

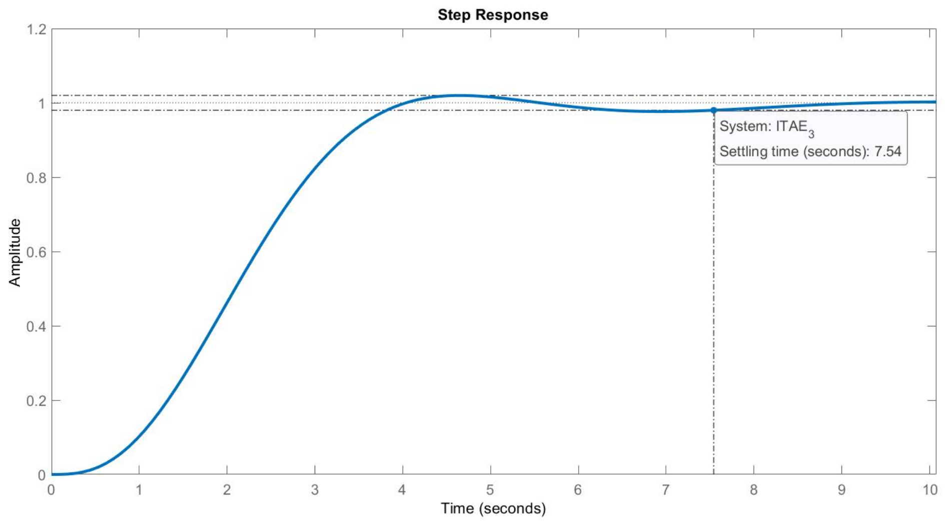


Figure 3.3: Normalized third order ITAE step response

$$\omega_n = \frac{T_{s,\ norm}}{T_{s,\ desired}} = \frac{7.54}{1} = 7.54 \tag{3.11}$$

Using the natural frequency found in (3.11), we obtain the third order ITAE transfer function in (3.12).

$$T_{ITAE}(s) = \frac{428.661}{s^3 + 13.195s^2 + 122.231s + 428.661} \tag{3.12}$$

The next step is to find the optimal forward path transfer function, $G_{fp}(s)$, and determine the resulting pole locations. We will then find the feedback gains necessary to place the closed-loop poles at the same location as the poles of the optimal forward path, $G_{fp}(s)$. Using the third order ITAE transfer function, the forward path is defined in (3.13).

$$\begin{aligned} G_{fp}(s) = \frac{T_{ITAE}(s)}{1 - T_{ITAE}(s)} &= \frac{428.661}{s(s^2 + 13.195s + 122.231)} \\ &= \frac{428.661}{s(s + 6.598 + j8.872)(s + 6.598 - j8.872)} \end{aligned} \tag{3.13}$$

As stated in the previous chapter, it is necessary to include an integrator in the design of the controller in order to eliminate steady-state error. This integrator is of the form $^1/_s$ in the Laplace domain. Inspection of (3.13) reveals an s term factored in the denominator. As a result, we will not have to consider this pole when using state variable feedback because it will be added with the inclusion of an integrator placed in series with $G_{sv\_fbk}(s)$.

The feedback gains are determined by finding the eigenvalues of the $(A + BK)$ matrix as shown in

(2.8). Using the *place* function in MATLAB, the state variable gain matrix necessary to locate the poles of the closed-loop system at the appropriate location is calculated. It should be noted that the *place* function in MATLAB returns the gain matrix necessary to move the eigenvalues of the $(A - BK)$ matrix. To compensate, the gains returned by MATLAB have been negated and the resulting gain matrix is shown in 3.14.

$$K = \begin{bmatrix} -0.5122 & -0.0133 \end{bmatrix} \tag{3.14}$$

The only remaining item to be designed is the pre-compensator. The purpose of the pre-compensator is to match the numerator of the plant to the numerator of the ITAE transfer function, $T_{ITAE}(s)$. Using equation (2.13), the transfer function of the plant with state variable feedback, $G_{sv\text{-}fbk}(s)$, is shown in (3.15).

$$\frac{Y(s)}{U(s)} = \frac{238.645}{s^2 + 13.195s + 122.231} \tag{3.15}$$

The denominator of the pre-compensator is set equal to the numerator in 3.15. Doing so effectively cancels the gain of $G_{sv\text{-}fbk}(s)$. We then match the gain of the ITAE transfer function by setting the numerator of the pre-compensator equal to the numerator of $T_{ITAE}(s)$. The resulting pre-compensator is shown in 3.16

$$\text{Pre-comp.} = \frac{428.661}{238.645} = 1.796 \tag{3.16}$$

Having determined the state variable feedback gains and pre-compensator required for the closed-loop system to match the ITAE transfer function, the design process is complete; however, before attempting to implement this controller on the E/Q testbed, it is first validated through simulation. A program written in MATLAB is used to simulate the response of the system. A step input with a magnitude of $\pi/2\ rad$ $(90°)$ is selected as the input. The system should output a final position of $\pi/2\ rad$ $(\approx 1.57\ rad)$ to the applied input. Additionally, the system should have a settling time of $1\ s$. The simulation results, shown in Fig. 3.4, confirm the controlled system has a settling time of $1\ s$. We are now ready to deploy the controller to the E/Q testbed.

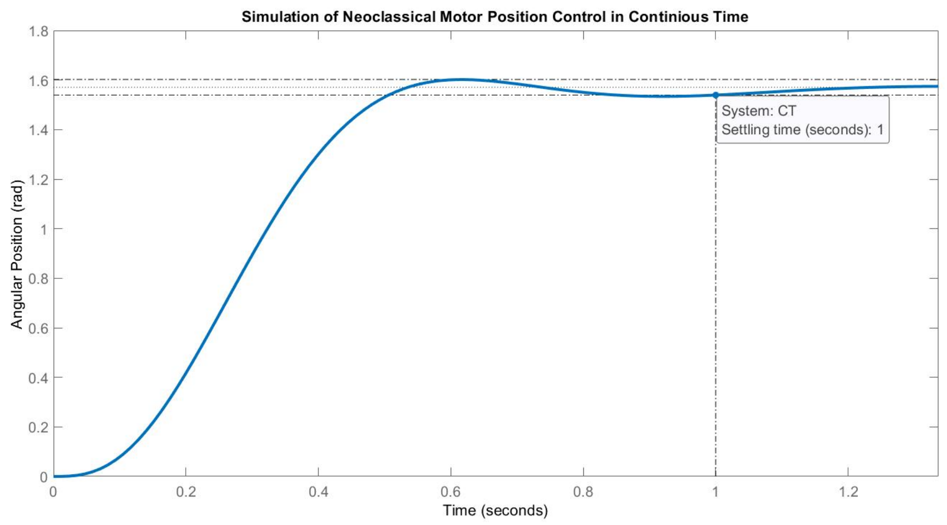
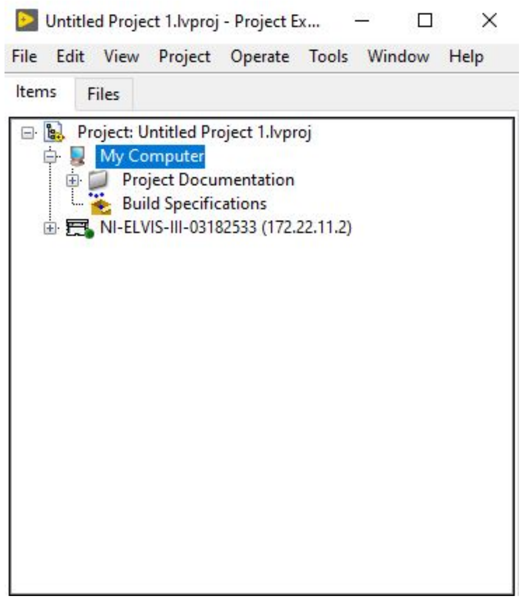


Figure 3.4: Simulation of neoclassical motor position control in continuous time

3.2.2 Deployment of Continuous Time Controller to Testbed

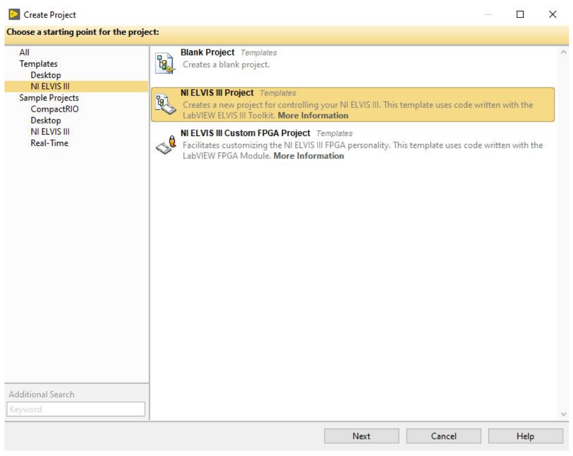
Having designed the controller, and validated it through simulation, the controller is now ready to be deployed to the E/Q testbed. The E/Q testbed first needs to be set up. Instructions for assembly and list of the needed software is provided in Appendix A.

Programming begins by connecting the ELVIS III to the host computer and launching the LabVIEW software. When prompted, a new project can be created using the NI ELVIS III template. The template automatically configures the settings necessary for the host computer to communicate with the ELVIS III. The only requirement from the user is to create a name for the project and define a directory in which the project will be stored. The windows used to create and configure a new project are shown in Fig. 3.5 and Fig. 3.6 respectively. Clicking finish opens the project explorer window shown in Fig. 3.7. From this window the user can create a new VI, which will be used to program the controllers. Two windows are activated when a new VI is created: the front panel and the block diagram. The front panel, which will be discussed in more detail later, is a graphical user interface that allows the user to interact with the VI. The block diagram is where the programming occurs.


Untitled Project 1.lvproj - Project Ex...
File Edit View Project Operate Tools Window Help
Items Files

Project: Untitled Project 1.lvproj
My Computer
Project Documentation
Build Specifications
NI-ELVIS-III-03182533 (172.22.11.2)

Figure 3.7: LabVIEW project explorer window



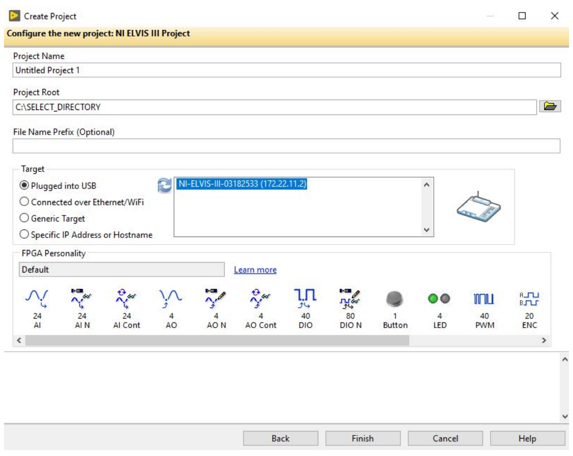
Choose a starting point for the project:

All
Templates
Desktop
NI ELVIS III
Sample Projects
CompactRIO
Desktop
NI ELVIS III
Real-Time

Blank Project *Templates*
Creates a blank project.

NI ELVIS III Project *Templates*
Creates a new project for controlling your NI ELVIS III. This template uses code written with the LabVIEW ELVIS III Toolkit. **More Information**

NI ELVIS III Custom FPGA Project *Templates*
Facilitates customizing the NI ELVIS III FPGA personality. This template uses code written with the LabVIEW FPGA Module. **More Information**



Configure the new project: NI ELVIS III Project

Project Name: Untitled Project 1

Project Root: C:\SELECT_DIRECTORY

File Name Prefix (Optional):

Target: [✓ Plugged into USB] Connected over Ethernet/WiFi Generic Target Specific IP Address or Hostname

NI-ELVIS-III-03182533 (172.22.11.2)

FPGA Personality: Default

Figure 3.5: LabVIEW create new project window

Figure 3.6: LabVIEW configure new project window

As mentioned in the previous chapter, LabVIEW uses a graphical programming environment. To create a program, the user chooses functions from a pallet. The functions are represented by icons that are placed on the block diagram. Most icons have been designed to resemble industry standard symbols. There is no code that needs to be written to create a controller. Instead, controllers are created by placing the icons into the block diagram window in such a way that they resemble the actual block diagram of the controller. LabVIEW then takes the block diagram and develops all the necessary code in the background.

The block diagram used to program the continuous time neoclassical controller is shown in Fig. 3.8. The LabVIEW block diagram closely resembles the general neoclassical block diagram shown in Fig. 2.8. With the exception of the motor and encoder VIs; all of the components used to create the LabVIEW implementation are available in the Control & Simulation pallet. The motor VI was created by Quanser and is used to generate an electrical signal within the ELVIS III and route it to the terminals of the motor. The signal is generated using one of ELVIS III's dedicated pulse width modulator (PWM) channels. Internally, the motor VI takes the input signal and determines an appropriate duty cycle for the PWM based on the magnitude of the regulated voltage. To determine the duty cycle, the motor VI normalizes its input to the maximum allowable magnitude of the regulated voltage. The result is a value that ranges from -1 to 1. The magnitude represents the duty cycle and the sign determines the direction of motor rotation. The voltage regulator ensures that the magnitude of an input to the motor VI never exceeds a maximum allowable level which ensures the calculated duty cycle cannot exceed 100%. The encoder VI, developed by Quanser, counts the pulses generated by the encoder and converts the result to an angular position expressed in radians.

All components used to implement the continuous time controller are contained within the Control & Simulation Loop. The Control & Simulation Loop is represented on the block diagram as the border shown in Fig. 3.8. Although it was not necessary for this design, components can be placed outside the loop structure. Any components placed outside of the loop structure will execute only once. The location of a component determines when that particular component is executed. For example, any component placed to the left of the loop structure will execute before the loop runs. The output from that component can then be passed into the loop structure before the loop begins to run. The components placed to the right of the loop structure will execute after the loop finishes running. In this case, when the loop has finished running, information will be passed out of the loop structure to the exterior functions that will then execute once.

In order to deploy continuous controller models to real-time targets, we must use the Control & Simulation Loop [15] because the ELVIS III is inherently a discrete controller itself. When a controller is deployed to ELVIS III, it is sent to a microprocessor which is a discrete device. The Control & Simulation Loop allows the user to design a continuous time controller; however, it is then discretized in the background before being deployed to the ELVIS III. To discretize the controller, LabVIEW converts everything on the block diagram to differential equations. Then, a numerical method, which is chosen by the user, is used to approximate a solution to the differential equations. The solution is then used to generate a control signal that is then delivered to the plant. While differential equations are indeed a continuous time representation, numerical approximation methods rely on time steps and are therefore discrete.

Once the controller has been designed on the LabVIEW block diagram, the only parameters that need to be considered are the numerical method to be used for discretization and the time step. There are several numerical methods that are available that fall into one of two categories: variable and fixed step size. Variable step size methods have the benefit of being highly accurate; however, this benefit comes at a computational cost. To achieve higher accuracy, an additional computation is carried out on each iteration to

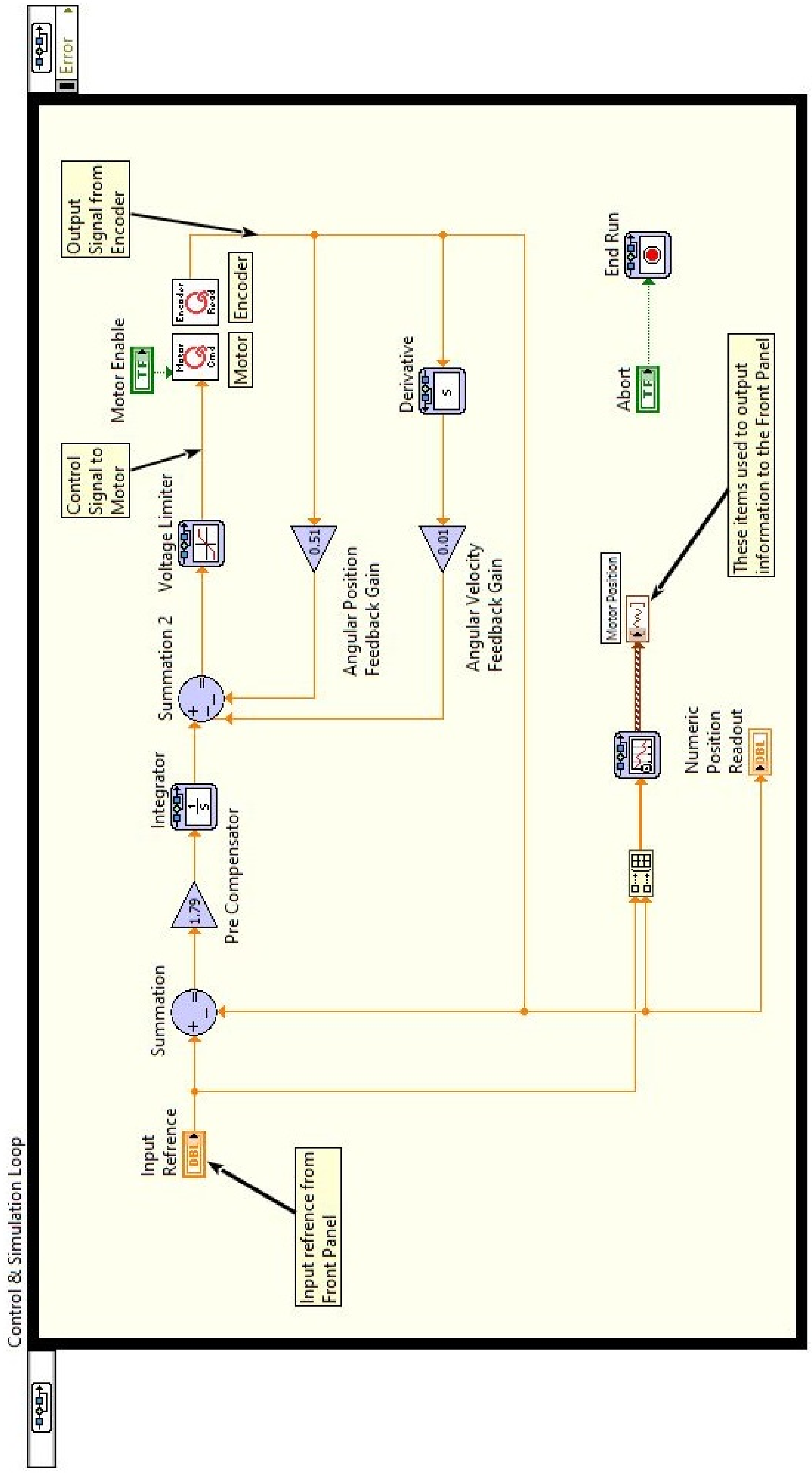

Control & Simulation Loop
Error
Input Refrence
Input refrence from Front Panel
Summation
Pre Compensator
1.79
Integrator
Summation 2
Voltage Limiter
Control Signal to Motor
Motor Enable
Motor
Encoder
Output Signal from Encoder
Angular Position Feedback Gain
0.51
Derivative
Angular Velocity Feedback Gain
0.01
Abort
End Run
These items used to output information to the Front Panel
Motor Position
Numeric Position Readout

Figure 3.8: LabVIEW block diagram of continuous time implementation

determine the magnitude of the maximum potential error resulting from the approximation. If the magnitude is greater than a preset threshold, then the step size is decreased, and the previous iteration is repeated until the potential error is within the acceptable tolerance. Because these computations are carried out on-line, the variable step methods are typically not an appropriate choice.

The alternative to using a variable step size solver are the methods that use a fixed step size. Unlike the variable methods, the fixed step size solvers do not calculate an error term at every iteration. Rather, as implied by the name, the step size remains fixed for all time regardless of the potential error in approximation. There are four fixed step size methods available for use which are Runge-Kutta orders one through four. The Runge-Kutta methods become more accurate as order increases. This added accuracy comes at the expense of additional computational complexity. Accuracy can also be increased by reducing the step size; however, there are limitations to how small we can make the step size. Computational resources place limits on how small we can make the step size for the simple fact that we cannot compute the next approximation until the current computation is complete. If the computational overhead of the numerical solver and complexity of a particular controller require a certain time to compute an approximation, then the step size cannot be chosen to be less than this time. Choosing a numerical method and step size depends on the complexity of the controller, speed of the response, and the desired precision of the output. Unfortunately, there is no single correct combination of numerical solver and step size for any particular controller. As a result, selecting the solver and step size relies on experience and trial and error.

After selecting the numerical solver (third order Runge-Kutta) and an appropriate step size (5 ms), the controller is ready to be deployed. Deployment can occur from either the front panel or block diagram by clicking the right arrow on the toolbar. When the right arrow is selected, LabVIEW converts the graphical program to code that is used by ELVIS III. Communication is then established between the host computer and ELVIS III and the program is sent. Communication remains active between the host computer and ELVIS III for the duration of the run-time, which allows the user to interact with the VI while it is active. The controller designed for this implementation requires the user to input a reference signal that is used to define the desired position of the motor shaft. The output is monitored on a plot located on the front panel.

The front panel of the controller deployed in continuous time is shown in Fig. 3.9. A reference signal is entered and sent to the running VI at roughly 2.8 seconds. The reference signal appears as the blue line on the plot and actual motor position is shown as the red line. The numerical position readout displays the current position of the motor shaft in radians. Data collected from the deployment is exported to MATLAB and plotted in Fig. 3.10 with the upper and lower 2% settling time bound included. The time scale has been adjusted so that time starts the instant the reference signal is delivered. The plot shows that the actual settling time is $T_s = 1.15\ s$, which is 15% greater than the desired settling time of $T_{s,\ desired} = 1\ s$. Further performance and design analysis is given in the results chapter.

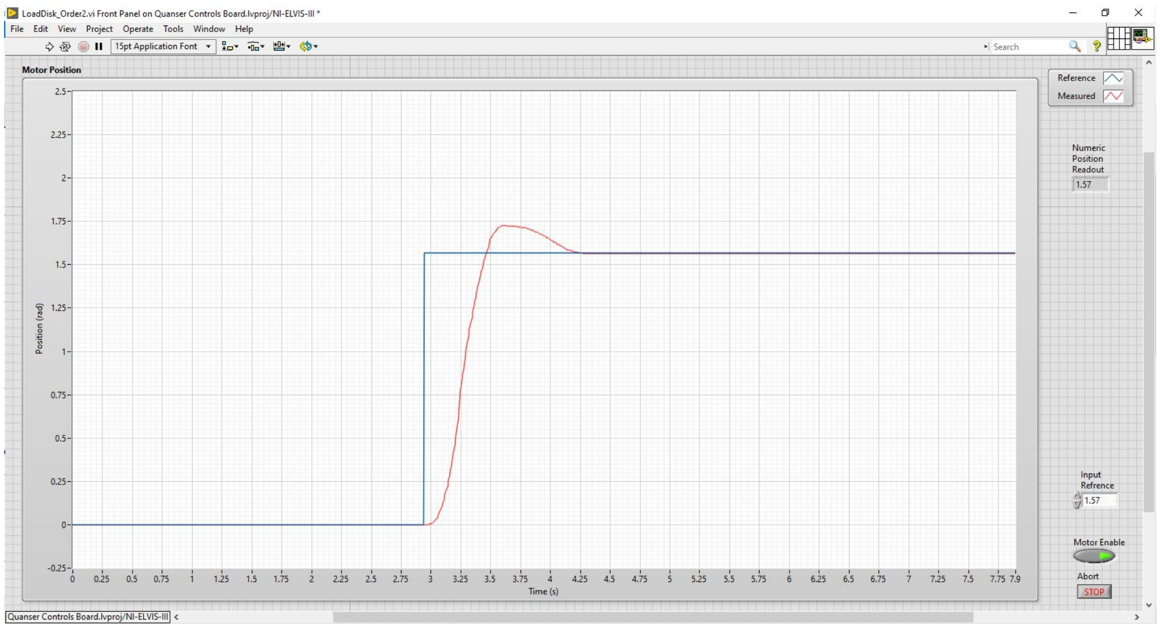

LoadDisk_Order2.vi Front Panel on Quanser Controls Board.lvproj/NI-ELVIS-III *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Motor Position
Reference
Measured
Numeric Position Readout
1.57
Position (rad)
Time (s)
Input Refrence
1.57
Motor Enable
Abort
STOP
Quanser Controls Board.lvproj/NI-ELVIS-III

Figure 3.9: Front panel of continuous time implementation

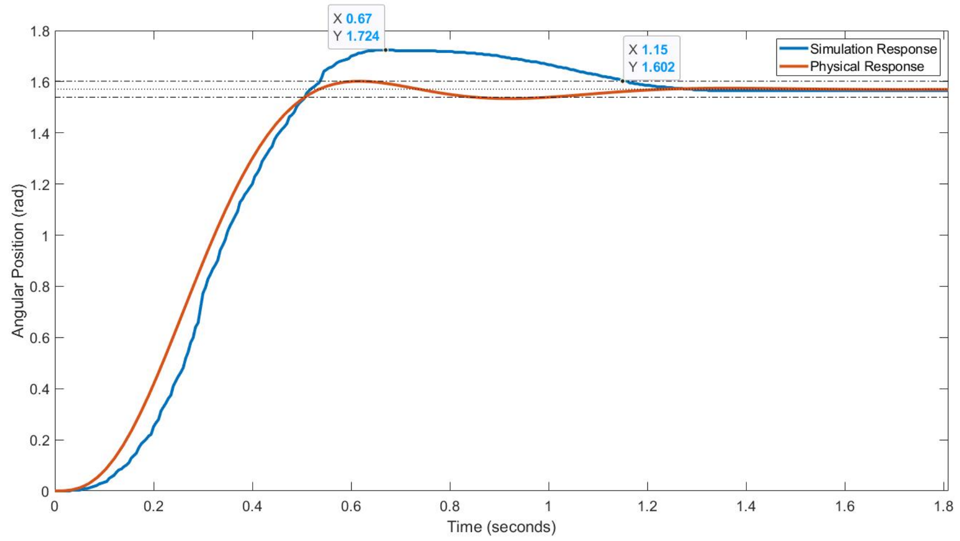


Figure 3.10: Continuous time neoclassical controller implementation

3.3 Discrete Time

The implementation of a discrete time controller designed under the neoclassical framework is considered in this section. A simulation of the controller is conducted using MATLAB to verify the design before the controller is deployed to the testbed.

3.3.1 Discrete Time Controller Design and Simulation

The first step in the design process is to select a model and an appropriate sampling time. Because the testbed is not capable of measuring the motor's current, the second order model, shown in (3.10), will be used to design the discrete time neoclassical controller. As a general guideline, the sampling frequency should be selected at 35 - 70 times greater than the 3db frequency, ω_b, of the closed loop system [16]. Because we know the closed-loop system will be designed to match the third-order ITAE transfer function, we can generate a Bode plot based on $T_{ITAE}(s)$ that will allow us to identify the 3db frequency. The only parameter needed to define the ITAE transfer function is the natural frequency. Selecting the desired settling time for this controller to be consistent with the continuous time controller, $T_{S,\ desired} = 1\ s$, allows us to use the ITAE transfer function defined in (3.12) which is repeated below for convenience.

$$T_{ITAE}(s) = \frac{428.661}{s^3 + 13.195s^2 + 122.231s + 428.661}$$

A Bode plot is generated in MATLAB and shown in Fig. 3.11. The plot indicates that the 3db frequency occurs at $\omega_b = 7.75\ {}^{rad}/{}_{s}$. Using the general guideline, the sampling frequency, ω_s, is selected to be 70 times greater than the 3db frequency (i.e. $\omega_s = 542.5\ {}^{rad}/{}_{s}$). The sampling frequency is converted to a sampling time of $T = 10\ ms$.

Having found an appropriate sampling time, the plant and third order ITAE transfer function are discretized using the zero-order hold method. The *c2d* command in MATLAB is used to carry out the discretization and the results shown in (3.17) and (3.18).

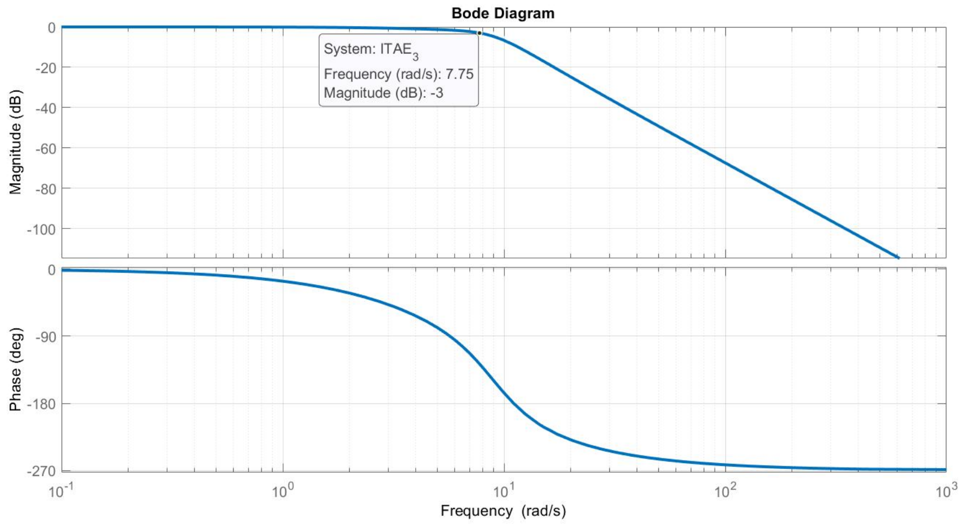


Figure 3.11: Bode plot of third order ITAE transfer function

$$\begin{bmatrix} \theta_m[k+1] \\ \omega_m[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0.0095 \\ 0 & 0.9046 \end{bmatrix} \begin{bmatrix} \theta_m[k] \\ \omega_m[k] \end{bmatrix} + \begin{bmatrix} 0.0115 \\ 2.2708 \end{bmatrix} v_m[k]$$

$$y[k] = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta_m[k] \\ \omega_m[k] \end{bmatrix} \tag{3.17}$$

$$T_{ITAE}(z) = \frac{10^{-5}(6.911z^2 + 26.74z + 6.469)}{z^3 - 2.865z^2 + 2.742z - 0.8764} \tag{3.18}$$

The next step is to determine the optimal forward path, $G_{fp}(z)$, using the discretized third order ITAE transfer function. The calculation is shown in (3.19).

$$G_{fp}(z) = \frac{T_{ITAE}(z)}{1 - T_{ITAE}(z)} = \frac{10^{-5}(6.911z^2 + 26.74z + 6.469)}{z^3 - 2.865z^2 + 2.741z - 0.8764}$$

$$= \frac{10^{-5}(6.911z^2 + 26.74z + 6.469)}{(z-1)(z - 0.9325 \pm j0.0827)} \tag{3.19}$$

We now determine the feedback gains necessary to match the system poles to the poles of $G_{fp}(z)$. Before going further, it is necessary to recall from Chapter 2 that the discrete time integrator takes the form $T/(z-1)$ where T is the sampling time. Inspection of (3.19) reveals a $z - 1$ term in the denominator which will be accounted for when we add the integrator to the system. Therefore, it is not necessary to consider this pole when determining the feedback gains. The *place* command in MATLAB is again used to determine the appropriate feedback gain matrix K, making the previously described adjustment to the sign of the gain. The resulting gain matrix is shown in (3.20).

$$K = \begin{bmatrix} -0.5126 & -0.0149 \end{bmatrix} \tag{3.20}$$

The final step is to determine the pre-compensation needed to match the numerator of the plant to the numerator of the ITAE transfer function $T_{ITAE}(z)$. This is achieved by first setting the denominator of the pre-compensator equal to the numerator of the plant. This allows for the cancellation of any zeros that may be present and removes any gain. We then set the numerator of the pre-compensator equal to the numerator of the ITAE transfer function $T_{ITAE}(z)$. MATLAB is used to convert the state space representation of the plant in (3.17), to a transfer function which is shown in (3.21).

$$G_{sys}(z) = \frac{10^{-3}(11.54z + 11.16)}{z^2 - 1.905z + 0.9046} \tag{3.21}$$

The pre-compensator can now be determined and is shown in (3.22).

$$\text{Pre-comp.} = \frac{10^{-2}(6.911z^2 + 26.74z + 6.469)}{11.54z + 11.16} \tag{3.22}$$

Before deploying the controller to the E/Q testbed, we validate the design using simulation. MATLAB is used to create a simulation to a step input with a magnitude of $\pi/2\ rad\ (90°)$. The resulting plot, Fig. 3.12, shows that the system reaches the desired output of $\pi/2\ rad\ (\approx 1.57\ rad)$ and has a settling time of $1s$. It is now appropriate to deploy the controller to the testbed.

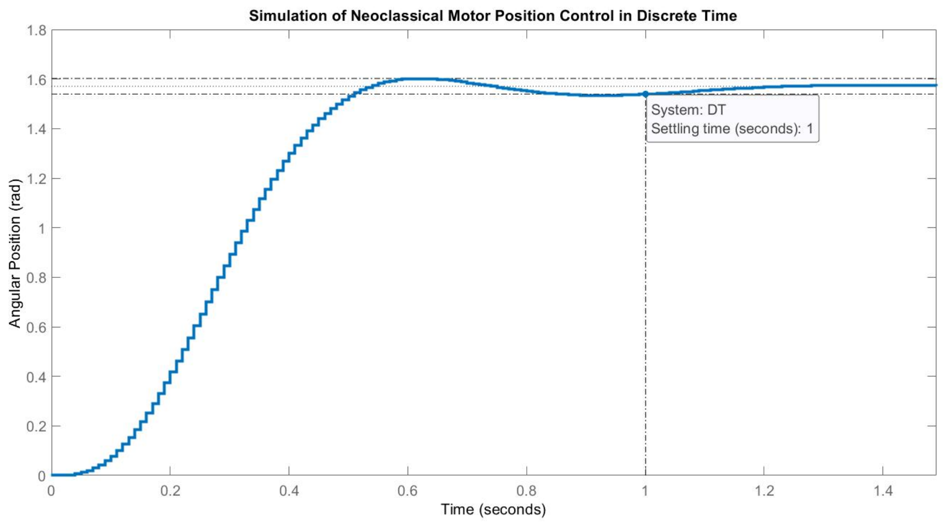


Figure 3.12: Simulation of neoclassical motor position control in discrete time

3.3.2 Deployment of Discrete Time Controller to Testbed

Discrete time neoclassical controller deployment begins with the creation of a new LabVIEW project and VI. The process is initially identical to the procedure used in continuous time. The major difference is with regard to the loop structure. With the continuous time implementation, it was necessary to use the Control & Simulation loop so that LabVIEW could discretize the design in the background. The discrete implementation is not restricted in the same way because the controller has already been discretized. As a result, we can use a different loop structure that does not rely on numerical approximations.

The loop structure that was chosen for the discrete time controller deployment is called a Timed Loop.

The Timed Loop iteratively executes instructions contained within its structure in pre-defined time intervals. The execution of instructions is carried out primarily from left to right; however, execution of any particular function will not take place until all necessary parameters and variable values are available.

The block diagram of the discrete implementation is shown in Fig. 3.13. Aside from the loop structure itself, there are only three differences between the continuous and discrete implementations. The first difference is the location of the encoder. In the continuous case, it did not matter where this was placed; however, in the discrete case this is somewhat important because the Timed Loop handles instruction from left to right. Since we need to know the initial conditions of the system; we ensure the encoder is the first function to run by placing it far to the left within the timed loop structure. There is an alternative approach that would allow us to place the encoder on the right hand side of the loop structure that will be discussed when we add an observer to the design. The next difference is the appearance of the summing junctions. The summing junctions found on the Control & Simulation pallet were not designed to be used within the timed loop structure. Instead, the timed loop structure uses the mathematical operators found on the Numeric pallet. These mathematical operators are essentially identical to the summing junctions found on the Control & Simulation pallet with the only difference being the number of inputs available. This may result in the need to cascade two, or more, mathematical operators. The final difference is the voltage limiter. This function, like the summing junction, is also not able to be used outside the Control & Simulation loop. The alternative is to create an equivalent using case structures. The case structures use logic to make decisions. In this case, if the signal is above (or below) a certain level then it is held at the maximum (or minimum) allowable value.

Once the controller has been designed, the only configuration that is required is loop timing. Loop timing is equivalent to iteration time step, which is nothing more than the sample time that was used to design the controller. The iteration time should be set to the sampling time used to design the controller, $10ms$ in this case.

The front panel of the implemented discrete time controller is shown in Fig. 3.14. The reference input is represented by the blue line on the plot and the actual position is represented by the red line. An reference input of ${}^{\pi}/_{2}\ rad$ is applied at approximately 3.4 seconds. Data collected from the deployment is exported to MATLAB and plotted in Fig. 3.15 with the upper and lower 2% settling time bound included. The time scale has been adjusted so that time starts the instant the reference signal is delivered. The plot shows that the actual settling time is $T_s = 1.15\ s$ which is 15% greater than the desired settling time of $T_{s,\ desired} = 1\ s$. It is noted that this settling time agrees exactly with the continuous time implementation. Further performance and design analysis is given in the results chapter.

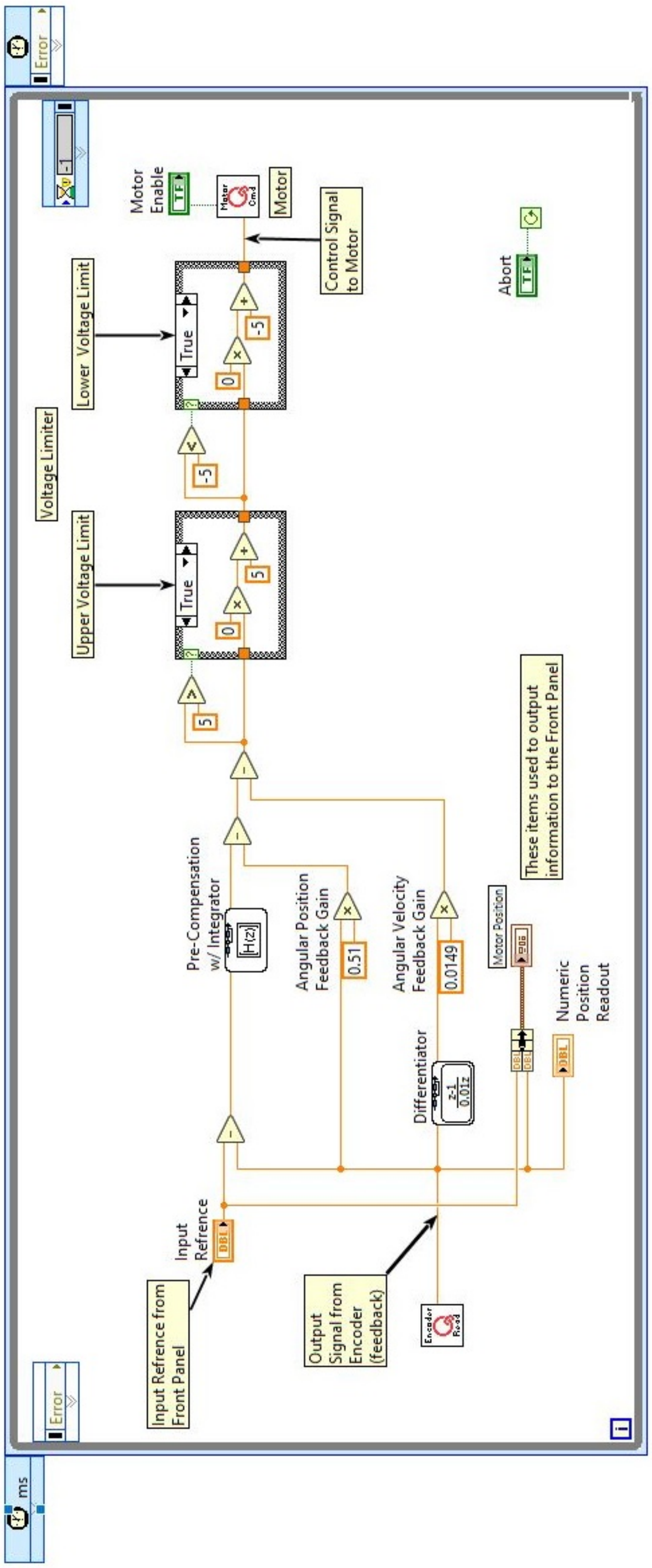


Figure 3.13: LabVIEW block diagram of discrete time implementation

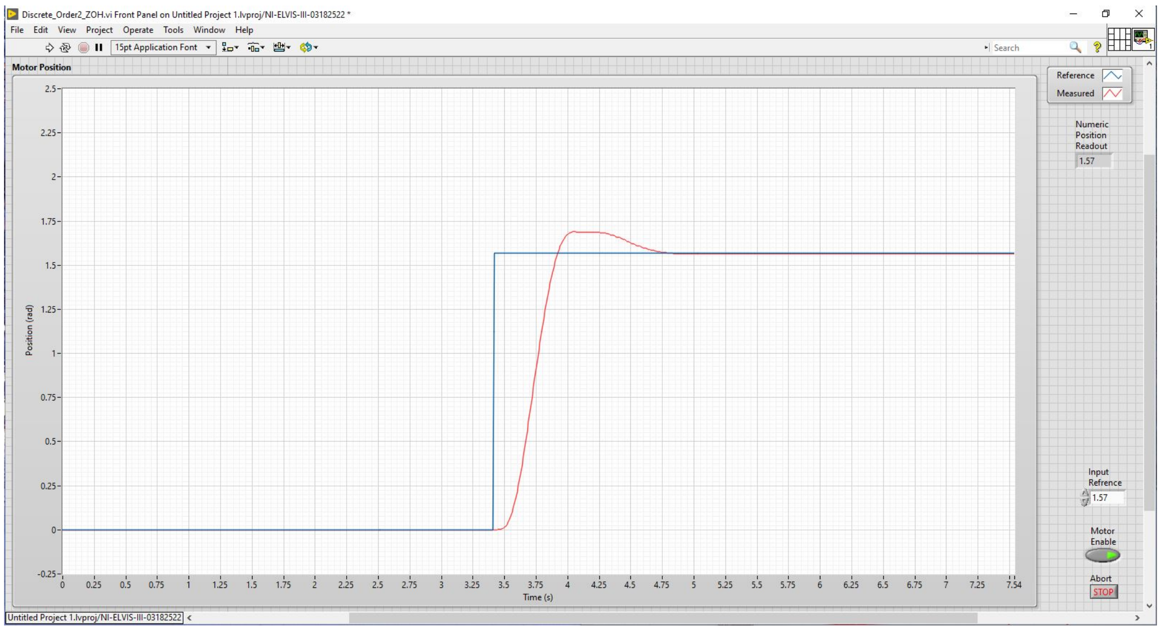

Discrete_Order2_ZOH.vi Front Panel on Untitled Project 1.lvproj/NI-ELVIS-III-03182522 *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Motor Position
Position (rad)
Time (s)
Reference
Measured
Numeric Position Readout
1.57
Input Refrence
1.57
Motor Enable
Abort
STOP
Untitled Project 1.lvproj/NI-ELVIS-III-03182522

Figure 3.14: Front panel of discrete time implementation

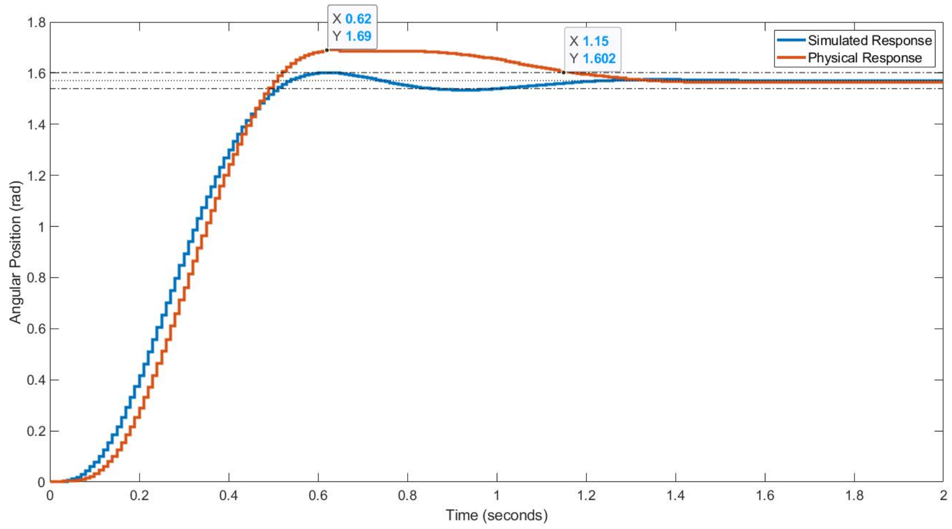


Figure 3.15: Discrete time neoclassical controller implementation

3.4 Discrete Time with Observer

Up until now, a second order model has been used to design the controllers deployed to the testbed. The second order model was necessary because one of the states of the original, third order, model was the motor's current, which we are not able to measure. In this section, we implement an observer that is used to estimate the value of the un-measurable state. The observer is a second system that works with the controlled system to estimate the states of the controlled system. While the observer runs independently of the controlled system, it is dependent on the input and output of the system. This section begins with the design of the neoclassical controller using the third order model in (3.7). Next, the observer is designed using the third order ITAE transfer function to determine the pole locations. Simulation results are provided before the controller is deployed to the testbed.

3.4.1 Discrete time Controller/Observer Design and Simulation

A block diagram of a discrete time system with observer designed using the neoclassical framework is shown in Fig. 3.16 where $\hat{X}(z)$ is the state estimate. To complete the design, we need to determine the unknown components in the block diagram which are the state variable gain matrix, K, the observer gain matrix, L, and the pre-compensator, $P.C.$ Before any of these components can be determined, we must first define an appropriate sampling time.

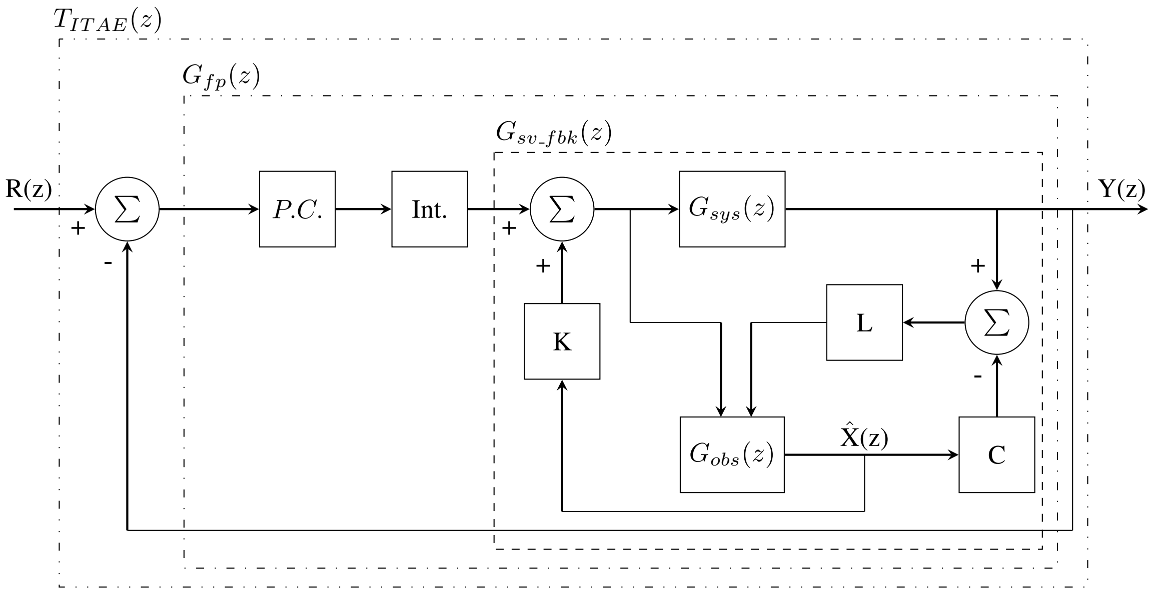


Figure 3.16: Block diagram of a discrete neoclassical controlled system with observer

The sampling time will be determined from the closed-loop response of system. Recall that the closed-loop system will be designed to match the ITAE transfer function that is an order higher than that of the plant. Because the plant is third order, we will need to use the fourth order ITAE transfer function. To find the fourth order ITAE transfer function, we need determine the required natural frequency of the controller. Using the same procedure as earlier, we find the settling time of the normalized fourth order ITAE transfer function to an applied step input. From Fig. 3.17, the normalized settling time is found to be

$T_{s,\ norm} = 4.51\ s$.

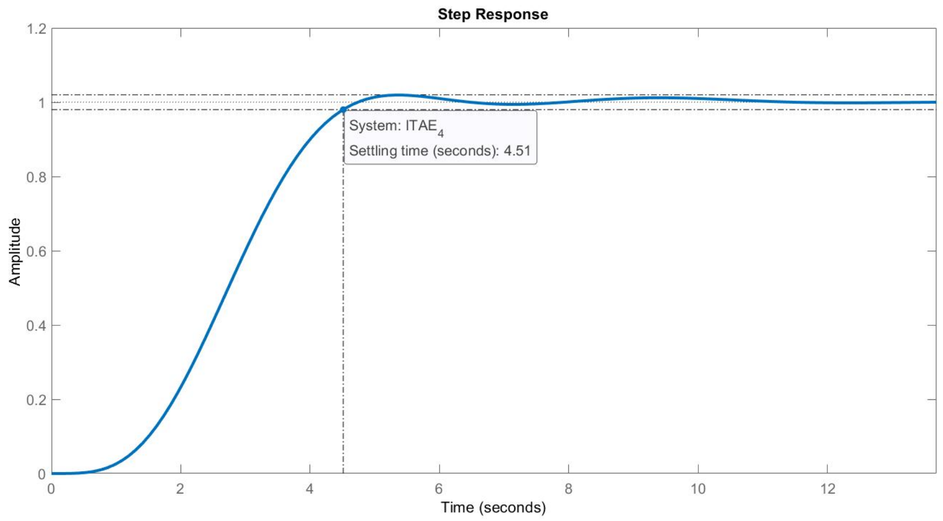


Figure 3.17: Normalized fourth order ITAE step response

We again choose the desired settling time to be $T_{s,\ desired} = 1\ s$. The natural frequency of the controller is then found using (2.4).

$$\omega_{n,\ ctrl} = \frac{T_{s,\ norm}}{T_{s,\ desired}} = \frac{4.51}{1} = 4.51 \tag{3.23}$$

which allows us to define the fourth order ITAE transfer function as;

$$T_{ITAE\_4}(s) = \frac{413.7}{s^4 + 9.471s^3 + 69.16s^2 + 247.7s + 413.7} \tag{3.24}$$

Using (3.24), we can generate a Bode plot using MATLAB that will allow us to select an appropriate sampling time. The Bode plot is shown in Fig. 3.18 where the 3db frequency is found to be $\omega_b = 4.02\ {}^{rad}\!/_{s}$. Like before, we define the sampling frequency to be 70 times greater than the 3db frequency. Therefore, the natural frequency is $\omega_s = 281.4\ {}^{rad}\!/_{s}$. Conversion of the sampling frequency yields a sampling time of $T = 20\ ms$

We now discretize the plant using the zero-order hold method at a sampling time of $T = 20\ ms$. The resulting state space representation and transfer function are shown in (3.25) and (3.26) respectively.

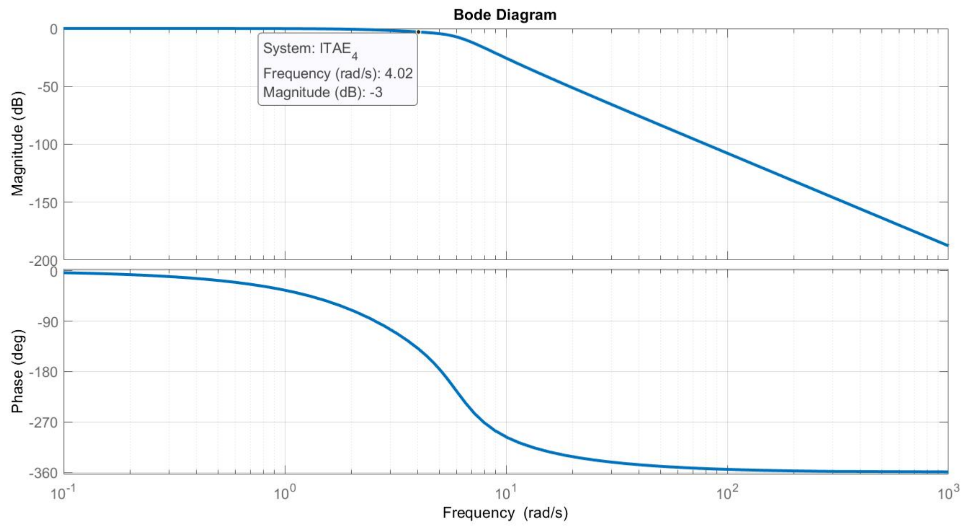


Figure 3.18: Bode plot of fourth order ITAE transfer function

$$\begin{bmatrix}\theta_m[k+1]\\ \omega_m[k+1]\\ i_m[k+1]\end{bmatrix} = \begin{bmatrix}1 & 0.0181 & 0.0050\\ 0 & 0.8193 & 0.2271\\ 0 & -0.0041 & -0.0011\end{bmatrix}\begin{bmatrix}\theta_m[k]\\ \omega_m[k]\\ i_m[k]\end{bmatrix} + \begin{bmatrix}0.0442\\ 4.3033\\ 0.0977\end{bmatrix} v_m[k]$$

$$y[k] = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}\theta_m[k]\\ \omega_m[k]\\ i_m[k]\end{bmatrix} \tag{3.25}$$

$$\begin{aligned} G_{sys}(z) &= \frac{0.04416z^2 + 0.04245z + 3.744e-06}{z^3 - 1.818z^2 + 0.8181z} \\ &= \frac{0.044156(z+0.9612)(z+8.821e-05)}{z(z-1)(z-0.8181)} \end{aligned} \tag{3.26}$$

Unfortunately, the system exhibited instability during implementation due to excessively large observer gains required by the design. Large observer gains are typically detrimental to stability because they can amplify measurement noise and saturate the controller. Euler's forward difference method of discretization (discussed in Chapter 2) was attempted but yielded even higher observer gains which, in-turn, caused similar instability issues when implemented. Pole-zero matching discretization method (also discussed in Chapter 2) was then attempted and showed promising results; however, the implemented system displayed some oscillatory behavior. To compensate, the sampling time was decreased from $T = 20\ ms$ to $T = 10\ ms$. It is permissible to decrease sampling time in order to improve performance provided the sampling time is not faster than the computational time required by the controller. The MATLAB *c2d* function is used to discretize the plant using pole-zero matching at a sampling time of $T = 10\ ms$. The resulting state space representation and transfer function are shown in (3.27) and (3.28) respectively. It is noted that the two zeros located at $z = -1$ lie on the unit circle. Canceling these zeros did not lead to instability during implementation.

$$\begin{bmatrix}\theta_m[k+1]\\ \omega_m[k+1]\\ i_m[k+1]\end{bmatrix} = \begin{bmatrix}0 & 1.38 & 1\\ 0 & 0.9045 & 1.38\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\theta_m[k]\\ \omega_m[k]\\ i_m[k]\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0.125\end{bmatrix} v_m[k]$$

$$y[k] = \begin{bmatrix}0.0455 & 0.0628 & 0.0455\end{bmatrix}\begin{bmatrix}\theta_m[k]\\ \omega_m[k]\\ i_m[k]\end{bmatrix} \tag{3.27}$$

$$G_{sys}(z) = \frac{0.00569(z+1)^2}{z(z-1)(z-0.9045)} \tag{3.28}$$

The next step is to discretize the fourth order ITAE transfer function in (3.24) and define the optimal forward path. Using the MATLAB *c2d* function with zero-order hold discretization at the sampling time of $T = 10\ ms$ yields (3.29).

$$T_{ITAE}(z) = \frac{10^{-7}(1.691z^3 + 18.25z^2 + 17.91z + 1.598)}{z^4 - 3.903z^3 + 5.716z^2 - 3.722z + 0.9096} \tag{3.29}$$

We now determine the forward path $G_{fp}(z)$;

$$\begin{aligned} G_{fp}(z) &= \frac{T_{ITAE}(z)}{1 - T_{ITAE}(z)} \\ &= \frac{10^{-7}(1.691z^3 + 18.25z^2 + 17.91z + 1.598)}{z^4 - 3.903z^3 + 5.716z^2 - 3.722z + 0.9096} \\ &= \frac{1.691 \times 10^{-7}(z+9.711)(z+0.9812)(z+0.09915)}{(z-1)(z-0.9483)(z-0.9775 \pm j0.0608)} \end{aligned} \tag{3.30}$$

Having found the optimal forward path, the feedback gain matrix, K, can be determined. Because we will include an integrator in the design, we do not need to consider the $(z-1)$ term in the denominator of (3.30) for pole placement. The *place* command in MATLAB is used to find the appropriate gains that will allow the poles of the closed-loop system to match the optimal forward path poles. It is again noted that the *place* function in MATLAB returns the gain matrix necessary to move the eigenvalues of the $(A - BK)$

matrix. Since we wish to move the eigenvalues of the $(A + BK)$ matrix, the gains returned by MATLAB have been negated. The resulting gain matrix is shown in (3.31).

$$K = \begin{bmatrix} 7.2771 & -11.1003 & 7.9874 \end{bmatrix} \tag{3.31}$$

The pre-compensator is determined next. As was the case for the previous controllers, the pre-compensator must cancel the numerator of the discretized plant and replace it with the numerator of the optimal forward path transfer function. Using (3.28) and (3.30) the pre-compensator is shown in (3.32).

$$\begin{aligned} \text{P.C.} &= \frac{1.6913 \times 10^{-7}(z + 9.711)(z + 0.9812)(z + 0.09915)}{0.00569(z + 1)^2} \\ &= \frac{2.9724 \times 10^{-3}(z + 9.711)(z + 0.9812)(z + 0.09915)}{(z + 1)^2} \end{aligned} \tag{3.32}$$

The next step is to design the observer that is used to estimate the states of the system. Most of the work has already been done because the system has already been discretized and the observer equations were derived last chapter in (2.11). The only work that needs to be done is to determine where the poles of the $(\boldsymbol{A} - \boldsymbol{LC})$ matrix should be located.

When an observer is incorporated into a neoclassical controller, the observer's pole locations are based on the ITAE transfer function. Up until now, we have used the ITAE transfer function that is an order higher than the plant to determine pole locations. In contrast, the observer bases its pole locations on the ITAE transfer function that is of the same order as the plant [6]. As a result, we will use the third order ITAE transfer function to determine appropriate locations for the observer poles. The desired settling time of the observer is the only consideration that needs to be made at this point. Because we need the state estimation to converge faster than the systems response, it is necessary to set the observer settling time to be faster than the controller. For this reason, the settling time is set to 10 times faster than the controller, or $T_{s,\ desired} = 0.1\ s$. Using the normalized third order settling time, the natural frequency is calculated in (3.33).

$$\begin{aligned} \omega_n &= \frac{T_{s,\ norm}}{T_{s,\ desired}} = \frac{7.54}{0.1} \\ &= 75.4 \end{aligned} \tag{3.33}$$

The third order ITAE transfer function used for observer pole placement can now be defined as:

$$T_{ITAE}(z) = \frac{4.9967(10^{-2})(z+2.592)(z+0.1991)}{(z-0.5863)(z^2-0.9356z+0.4558)} \tag{3.34}$$

The *place* function in MATLAB is used to determine the observer gain matrix which is shown in (3.35).

$$L = \begin{bmatrix} 0.0890 \\ 5.1694 \\ 1.1820 \end{bmatrix} \tag{3.35}$$

All necessary parameters required to construct the observer based neoclassical controller for the third order model have been determined. A simulation in MATLAB is used to verify the design before attempting to deploy to the testbed. The resulting plot, Fig. 3.19, shows that the system reaches the desired output of $\pi/2\ rad$ ($\approx 1.57\ rad$) and has a settling time of $1s$. It is now appropriate to deploy the controller to the E/Q testbed.

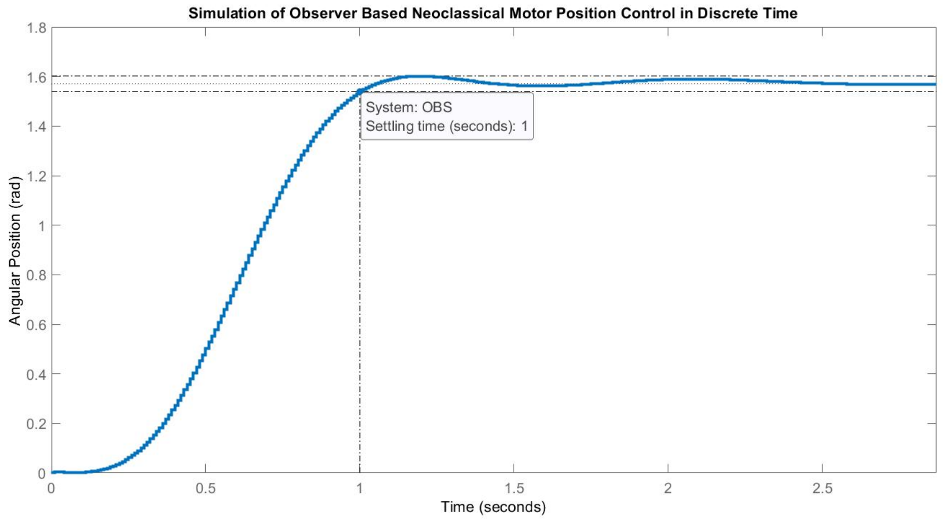


Figure 3.19: Simulation of observer based discrete time neoclassical motor position controller

3.4.2 Deployment of Discrete Time Controller/Observer to Testbed

The deployment process for the observer based discrete time design begins identically to the discrete controller discussed last section. We will use a timed loop structure and set the iteration period to match the sampling time of $T = 10\ ms$ which was used to design the controller. The only difference between the two implementations is the observer and the components required for the observer to function. The block diagram of the observer based discrete time controller is shown in Fig. 3.20. LabVIEW provides an observer VI to facilitate implementation of controllers that require the use of an observer. The observer VI is located near the center of the block diagram in Fig. 3.20. All individual components related to the observer based neoclassical controller implementation are discussed in detail.

The observer is designed based on the model of the plant. Therefore, we need to provide the observer VI with our model. Figure 3.21 shows how this can be accomplished. The CD construct state-space VI is used to create a state space model which is passed to the observer VI. Because anything located outside the loop structure executes only once, we define the state space model here. Doing so optimizes performance by preventing the model from being unnecessarily redefined on every iteration. If the system was time variant, then we may wish to define the state space model within the loop structure to accommodate changes to the system dynamics with respect to time. Before the timing loop begins, the model is defined and passed to the observer VI.

The observer requires two inputs; the control signal, $u[k]$, and the system output, $y[k]$. Additionally, the observer gain matrix, L, needs to be provided. Figure 3.22 shows how the control signal is brought to the observer. Because the observer VI requires this variable to be passed as an array. The data type is converted from a single 16-bit double to an array of 16-bit doubles. Since we only have a single control input the array has one dimension.

The measured system output, $y[k]$, is brought to the observer as shown in Fig 3.23. The output is obtained directly from the encoder and converted to a one dimension array. The observer gain is passed to the observer similar to the state space model as shown in Fig. 3.24. Because the gains are static they can be defined outside the loop structure.

The observer design is complete and all that remains is to apply state variable feedback to the estimated states. The method used to accomplish this is shown in Fig. 3.25. The observer outputs the state estimates in the form of an array. In order to access the individual states, we convert the array to a cluster. The cluster can then be un-bundled to allow access to the individual state estimates. Because our system is third order, we have three state estimates that occupy the first three elements respectively. All unused elements contain a null value. Having separated the estimated states, the feedback gain can be applied. They are then added to the system through a feedback node. The feedback node stores information from one iteration and uses it on the next. As a result, we need to define the initial conditions for the estimated state to the feedback node.

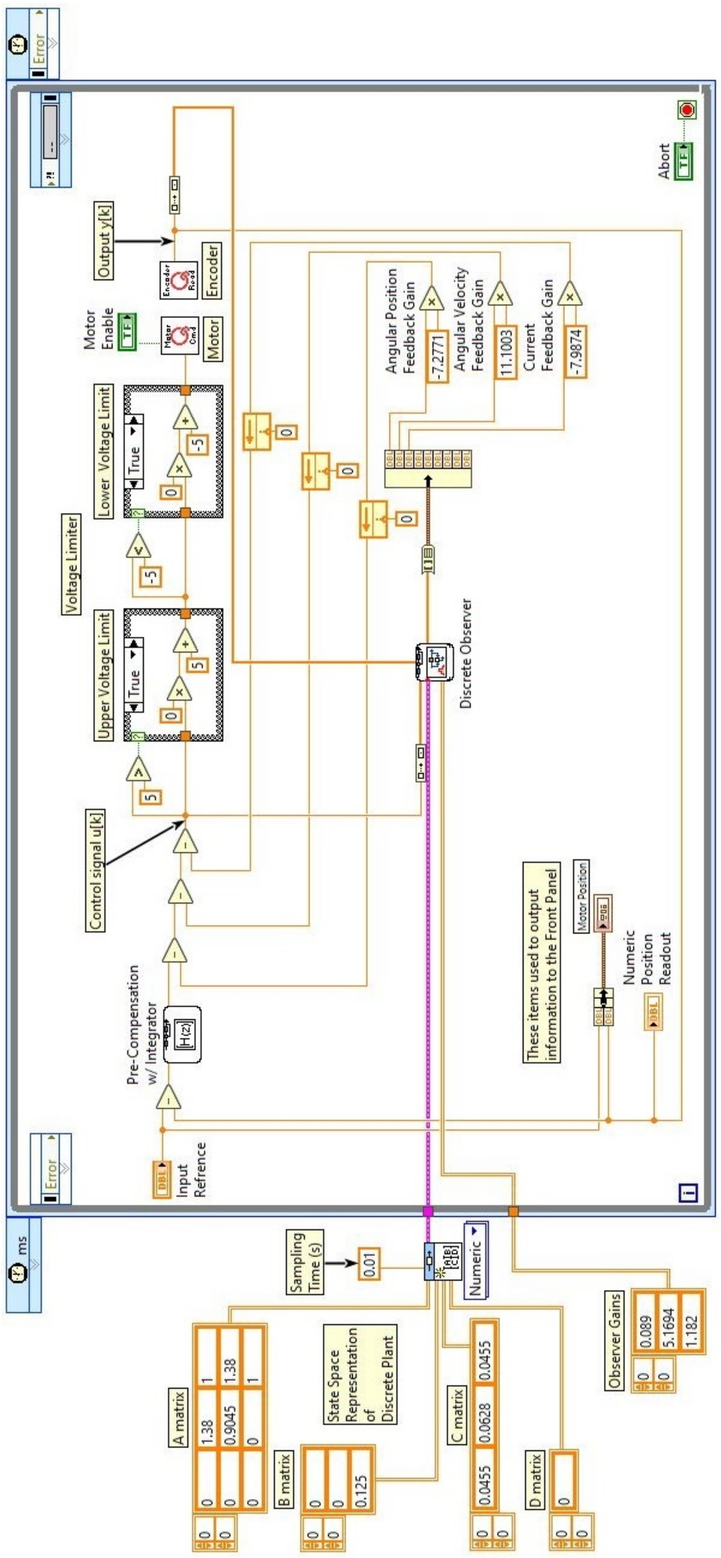


Figure 3.20: LabVIEW block diagram of observer based discrete controller

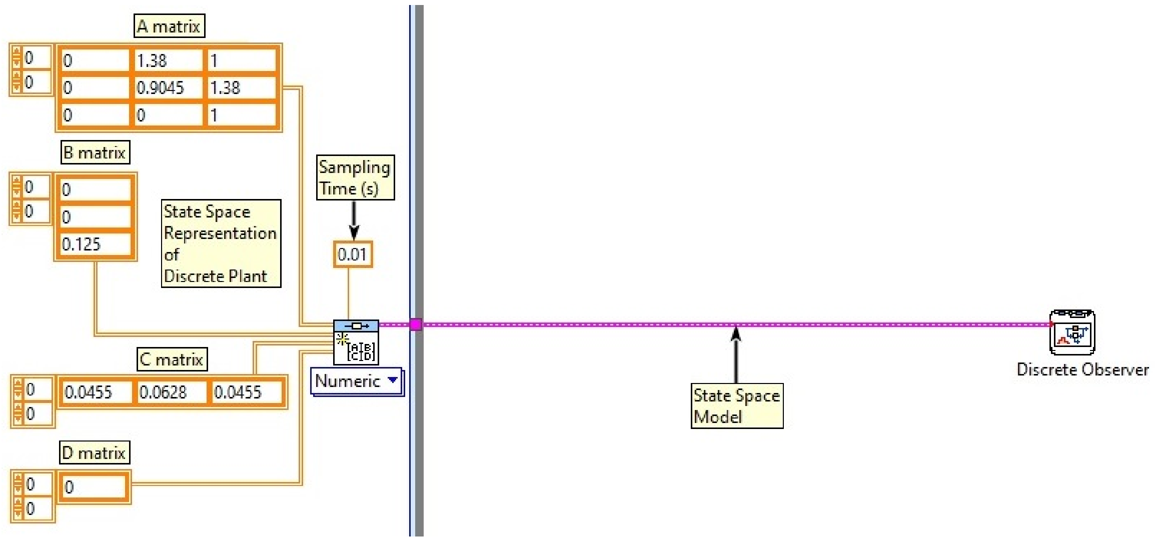


Figure 3.21: Entering state space model as parameter to observer

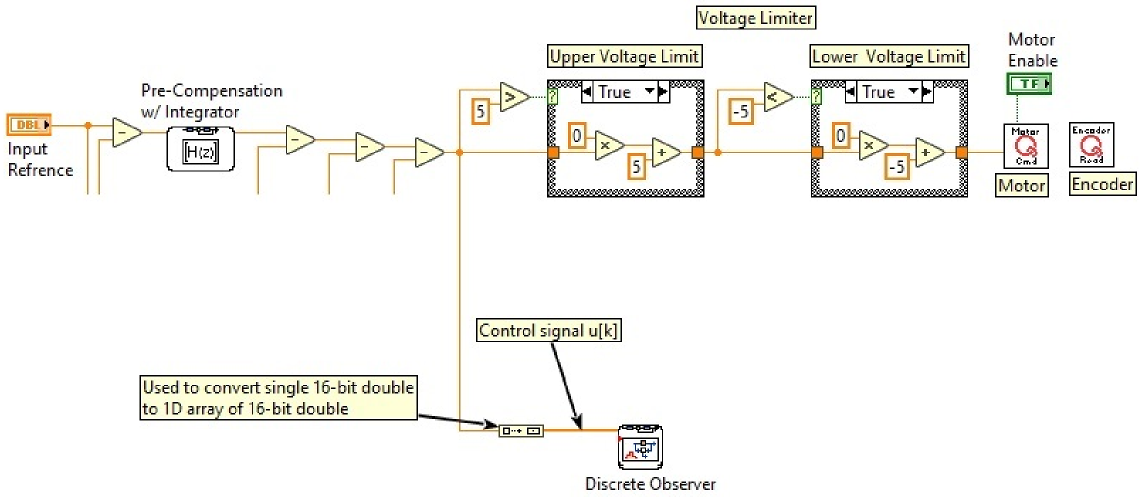


Figure 3.22: Control signal input to the observer

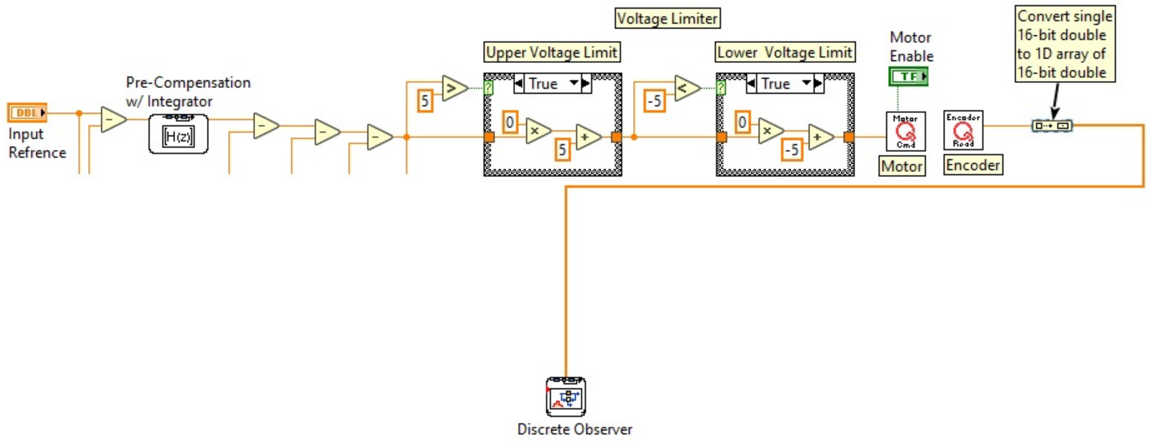


Figure 3.23: System output as an input to the observer

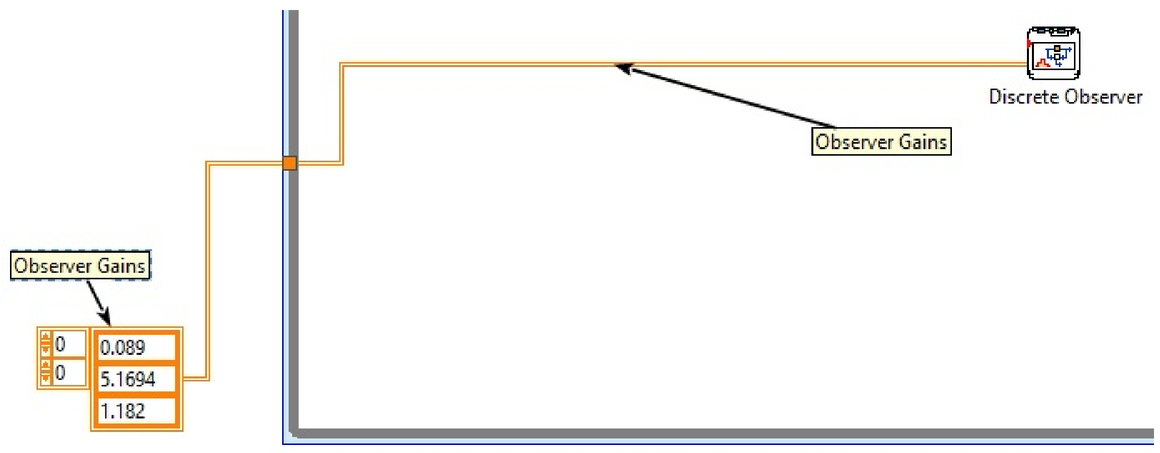
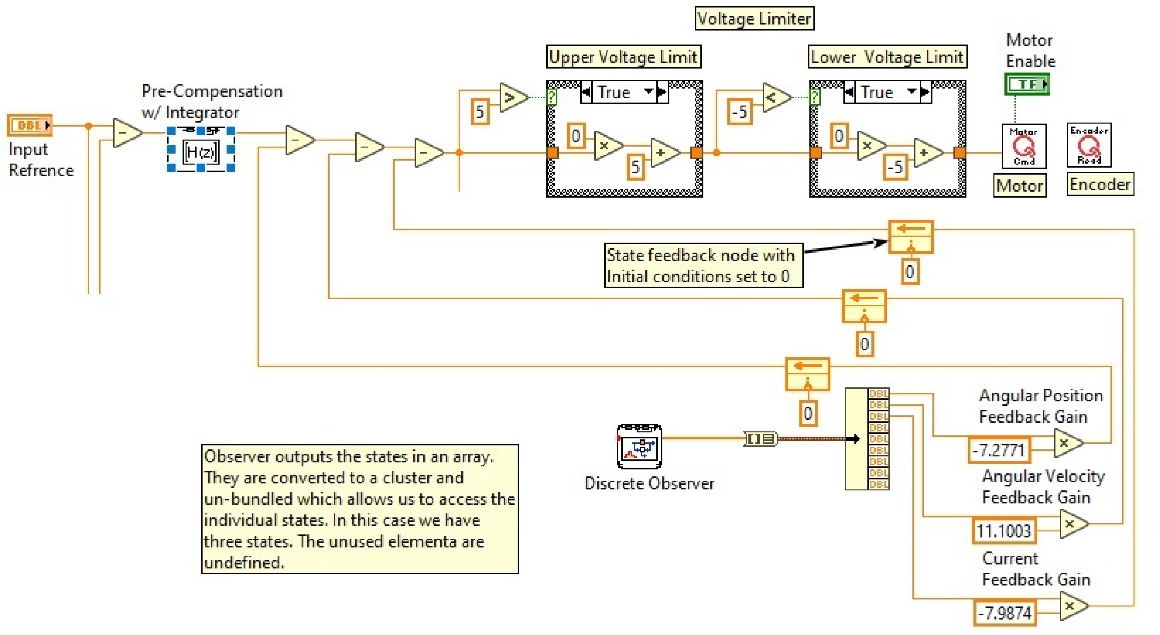


Figure 3.24: Gain input to the observer



Observer outputs the states in an array. They are converted to a cluster and un-bundled which allows us to access the individual states. In this case we have three states. The unused elementa are undefined.

Figure 3.25: Applying state variable feedback to estimated states

The front panel of the implemented observer based discrete time controller is shown in Fig. 3.26. The reference input is represented by the blue line on the plot and the actual position is represented by the red line. An input of $^{\pi}/_{2}\ rad$ radians is applied at approximately 3.4 seconds. Data collected from the deployment is exported to MATLAB and plotted in Fig. 3.27 with the upper and lower 2% settling time bound included. The time scale has been adjusted so that time starts the instant the reference signal is delivered. The plot shows that the actual settling time is $T_s = 1.05\ s$ which is 5% slower than the desired settling time of $T_{s,\ desired} = 1\ s$. Further performance and design analysis is given in the results chapter.

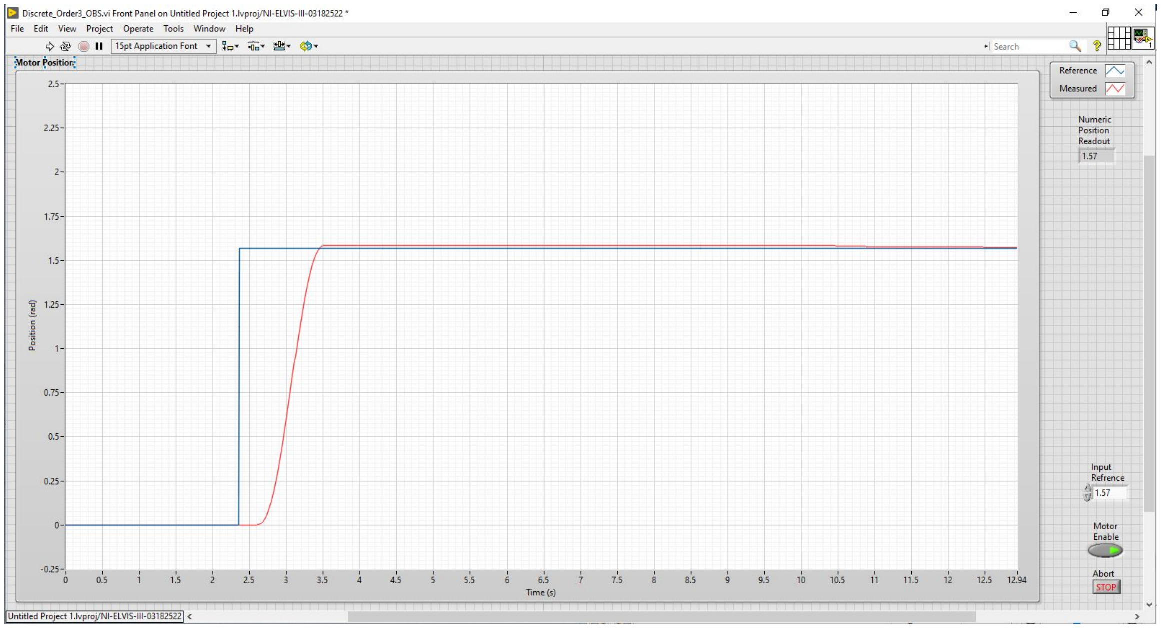

Discrete_Order3_OBS.vi Front Panel on Untitled Project 1.lvproj/NI-ELVIS-III-03182522 *
File Edit View Project Operate Tools Window Help
15pt Application Font
Motor Position
Position (rad)
Time (s)
Reference
Measured
Numeric Position Readout
1.57
Input Refrence
1.57
Motor Enable
Abort
STOP
Untitled Project 1.lvproj/NI-ELVIS-III-03182522

Figure 3.26: Front panel of observer based discrete time implementation

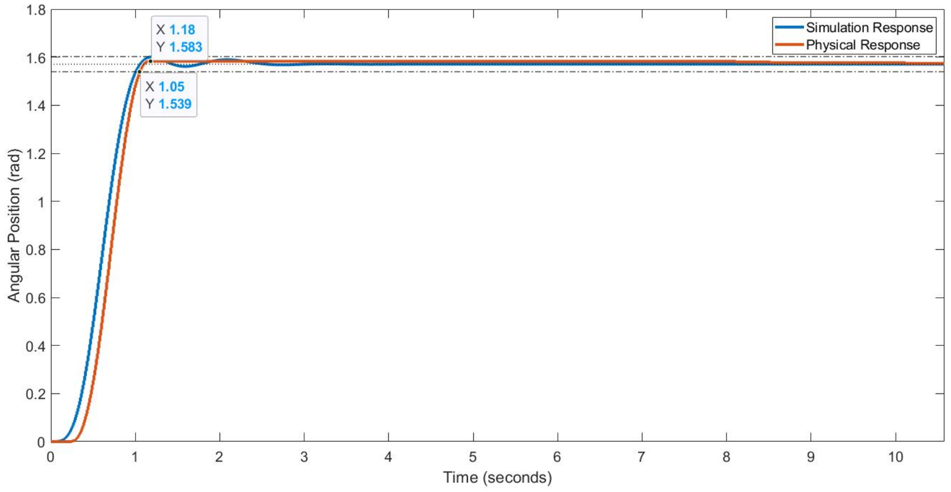


Figure 3.27: Observer based discrete time neoclassical controller implementation

3.5 PID Controller

A PID controller is a very well known control scheme that is commonly used in control systems. The controller is comprised of a proportional gain that is combined with integral and derivative gains which are typically cascaded with the plant. These gains are used to alter the poles and zeros of a closed-loop system. We will use the PID controller as a standard to compare against the neoclassical design technique. This section will focus on the design and implementation of the PID controller.

3.5.1 PID Controller Design and Simulation

Because the majority of the controllers in this work were designed in discrete time, we will choose to design a discrete PID. There are many methods that can be used to design a discrete PID controller. A common approach to discrete controller design in general is to first design the controller in the s-domain and then discretize the result. We will use this approach to design the discrete PID.

A block diagram of a PID controller in s-domain is shown in Fig. 3.28 where K_p, K_i, and K_d are the proportional, integral, and derivative gains respectively. From the figure, we can see that the PID does not rely on state variable feedback; unlike the controllers covered thus far. As a result, we have no need to measure the individual states of the system and will therefore use the third order model derived in (3.7) for our design.

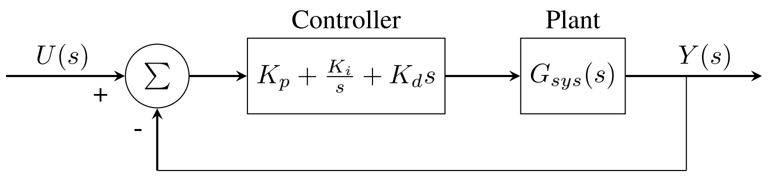
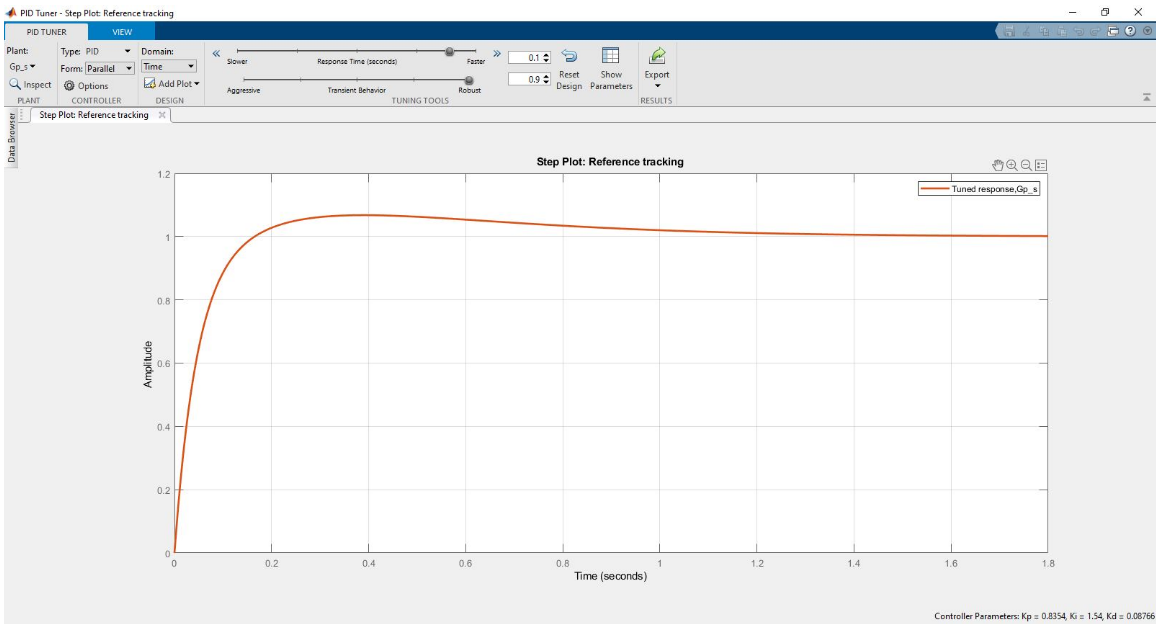


Figure 3.28: Block diagram of a PID controller in s-domain

The PID has been studied extensively thorough the years which has lead to the development of many different approaches to PID design. Some methods rely on analytical approaches while others, like the Ziegler-Nichols method, are performed in real-time with the plant. In either case, successive iterations, or tuning, is almost always required. Since ease-of-design is one of the comparisons we will eventually consider between the PID and neoclassical controllers, it follows that we should choose a simple approach to design this PID. For this reason we will use the PID tuner toolbox in MATLAB.

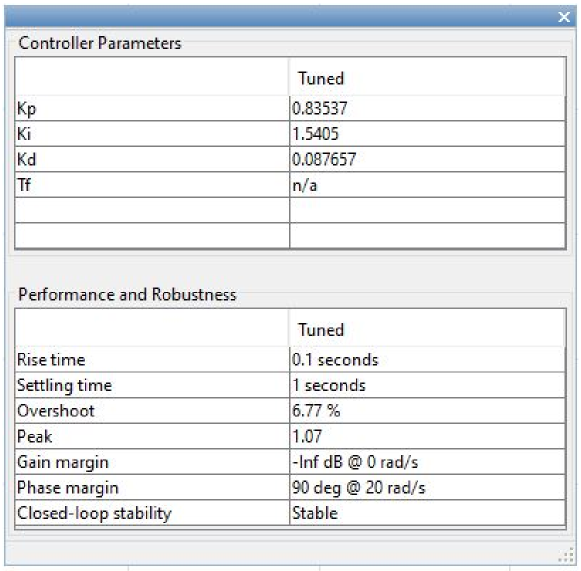
Design using the PID tuner begins by inputting the s-domain transfer function of the plant and selecting which variation of the PID controller is desired (i.e. P, PI, PD, PID). The PID tuner then simulates a closed-loop step response and displays the output. The gains are displayed together with performance measurements such as rise time, settling time, and percent overshoot. The closed-loop response is modified using two slide bars. One of the slide bars is used to control the speed of the response. The other slide bar allows the user to adjust the transient behavior of the response. In particular, the left extreme of the slide bar makes the response more aggressive to disturbance rejection whereas the extreme right of the slide bar makes the response more robust to parameter variation. The response plot, performance measurements, and gain values are updated instantly when a change is made to either of the slide bars. The ability to make adjustments to the response and instantly see the effect on the system makes this approach to PID design very efficient.

For our previous designs, the only criteria we were required to specify was the settling time. In every design we chose the settling time to be $T_{s,\ desired} = 1\ s$. We will therefore design our PID to this same specification by moving the response time slide bar until this settling time has been achieved. We will also choose our system to be highly robust because our model parameters were obtained from a generic data sheet and may have some variation between nominal and actual values. The main window used for the PID design is shown in Fig. 3.29 and the parameters and performance measures are shown in Fig. 3.30. Using the gains shown in Fig. 3.30 the PID controller, $G_c(s)$, is determined and shown in (3.36).



Controller Parameters: Kp = 0.8354, Ki = 1.54, Kd = 0.08766

Figure 3.29: PID design with MATLAB PID Tuner toolbox



Controller Parameters

	Tuned
Kp	0.83537
Ki	1.5405
Kd	0.087657
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.1 seconds
Settling time	1 seconds
Overshoot	6.77 %
Peak	1.07
Gain margin	-Inf dB @ 0 rad/s
Phase margin	90 deg @ 20 rad/s
Closed-loop stability	Stable

Figure 3.30: Controller parameters and performance

$$G_c(s) = K_p + \frac{K_i}{s} + K_d = 0.835 + \frac{1.504}{s} + 0.088s$$

$$= \frac{0.088s^2 + 0.835s + 1.504}{s} \tag{3.36}$$

The next step is to determine an appropriate sampling time and then discretize the plant and controller. Using the same methods as discussed in the design section of both discrete controllers, the sampling time is chosen to be $T = 10\ ms$. The plant is discretized using the zero-order hold method and is shown in (3.37).

$$G_{sys}(z) = \frac{0.044156(z + 0.9612)(z + 8.821e - 05)}{z(z-1)(z-0.8181)} \tag{3.37}$$

An issue occurs when we attempt to discretize the controller using the zero-order-hold method. The zero-order-hold method resulted in an unstable system. The forward difference and pole-zero matching methods both yielded a stable system whose simulated discrete time response was similar to that shown in Fig 3.29; however the transfer functions to both controllers are improper (i.e. contain more zeros than poles). Because improper transfer functions are not valid in LabVIEW, we cannot use these methods of discretization to deploy the controller to the E/Q testbed. The solution is to use the bilinear transformation.

The bilinear transformation is obtained from the relationship between z-domain and s-domain poles. As discussed in Chapter 2, s-domain poles can be mapped into the z-domain using $z = e^{sT}$ where s is an s-domain pole, z is a z-domain pole, and T is the discrete sampling time. If we rearrange this expression and consider a first-order approximation to the natural log, we arrive at the bilinear transformation as shown below.

$$z = e^{sT} \implies s = \frac{1}{T} ln(z) \approx \frac{2}{T}\left(\frac{z-1}{z+1}\right)$$

Using the bilinear transformation, the controller is defined in (3.38).

$$G_c(z) = \frac{18.38z^2 - 35.06z + 16.71}{z^2 - 1} \tag{3.38}$$

3.5.2 Deployment of PID Controller to the E/Q Testbed

The deployment of the PID controller to the E/Q testbed is very similar to both previous discrete deployments. We use a timed loop structure and set the sampling time to $T = 10\ ms$. The block diagram of the controller is shown in Fig 3.31. The PID controller is less complicated than any of the prior designs because there is no need to measure each state variable and feed them back to the system with an applied gain. The entire controller is contained within a single block into which we have entered the discretized PID transfer function.

The controller is deployed to the E/Q testbed and the response is recorded on the front panel which is shown in Fig. 3.32. It is clear that the settling time of this response is greater than the one second which was specified. Because this design does not meet the specification we will manually tune the gains until we achieve the desired response.

Manual PID tuning is often required to obtain the desired response from physical hardware. The goal is to make adjustments to the controller gains in order to meet design specification. Manual tuning can be difficult because an individual gain can be adjusted to provide a positive impact to one performance characteristic while having a negative influence an another. For example, increasing a derivative gain will reduce settling time; however, too much of an increase may amplify noise which could result in instability. Additionally, increasing proportional gain can reduce settling time but may increase percent overshoot. For this reason, it can be difficult to predict the impact an adjustment to a single controller gain may have on the response of a closed-loop system. This difficulty is compounded when considering how to adjust multiple gains. As a result manual tuning can be an exhaustively iterative process.

Using the controller gains obtained from the PID tuner toolbox (Fig. 3.30) as a starting point, the gains are adjusted directly on the E/Q testbed and deployed without prior simulation. After several deployments, a PID controller that meets the settling time specification of $T_{s,\ desired} = 1\ s$ is found. To arrive at this controller we increase the proportional gain to $K_p = 3.75$ and the derivative gain to $K_d = 0.15$. The integral gain remains unchanged. The controller is again discretized using the bilinear transformation and is shown in (3.39).

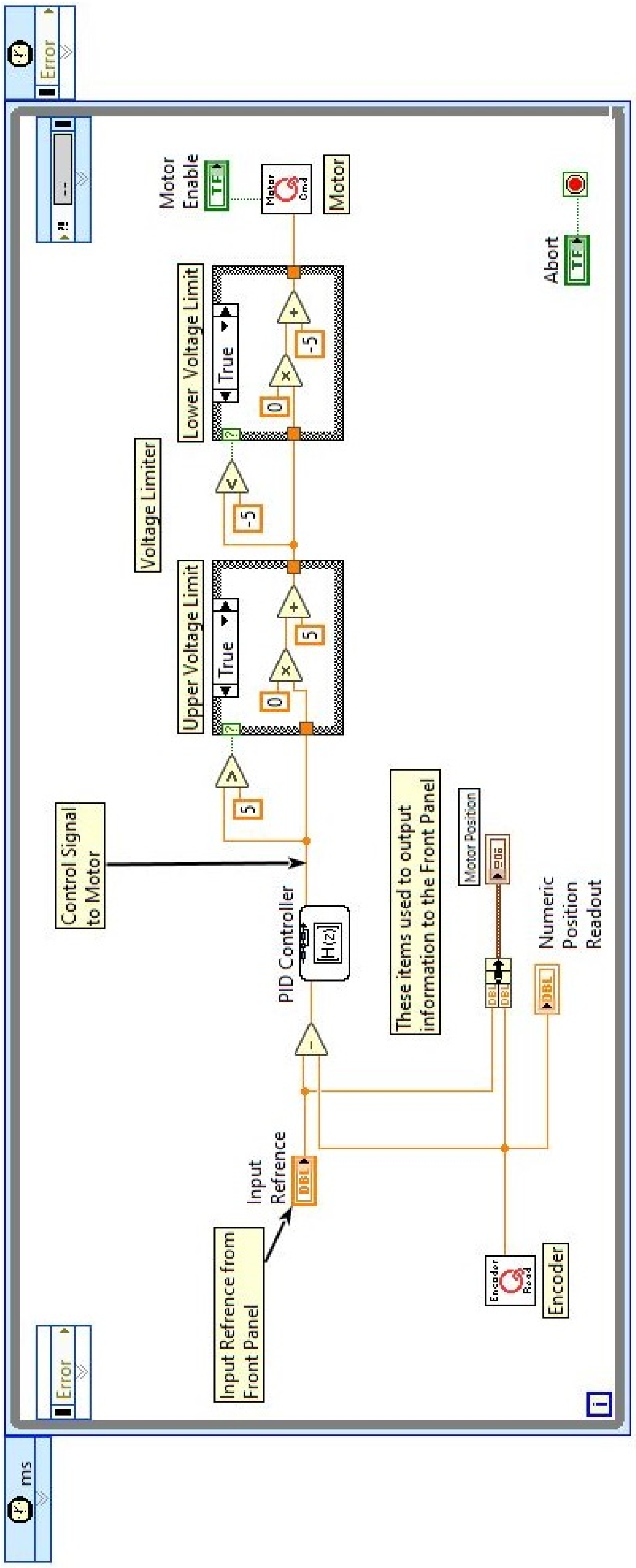


Figure 3.31: LabVIEW block diagram of PID controller

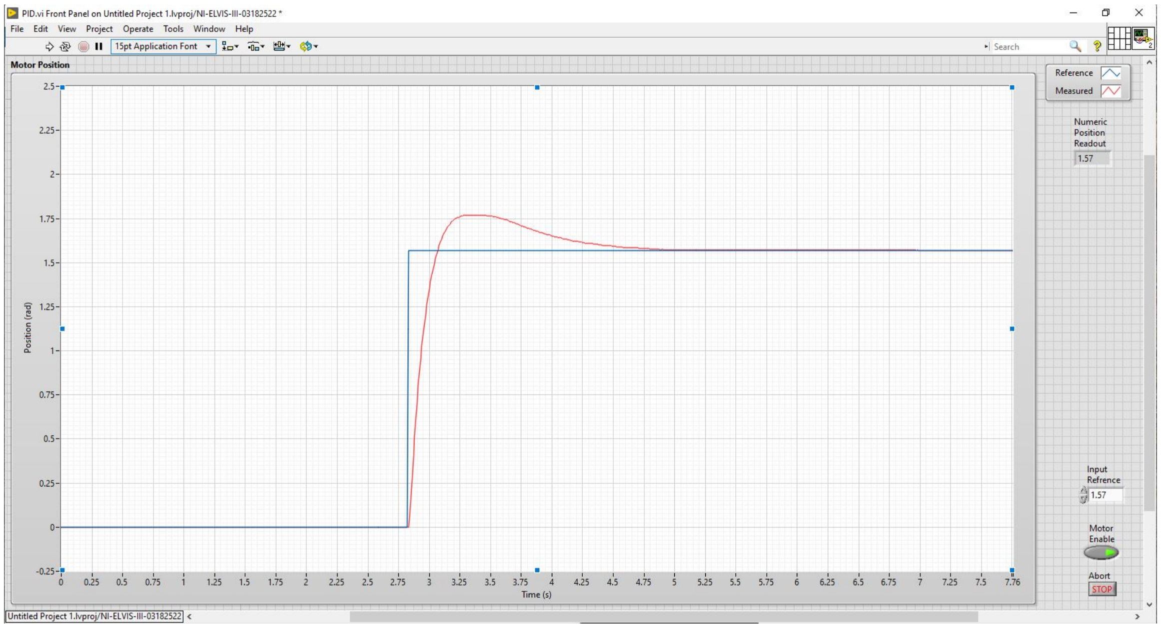

PID.vi Front Panel on Untitled Project 1.lvproj/NI-ELVIS-III-03182522 *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Motor Position
Reference
Measured
Numeric Position Readout
1.57
Position (rad)
Time (s)
Input Refrence
1.57
Motor Enable
Abort
STOP
Untitled Project 1.lvproj/NI-ELVIS-III-03182522

Figure 3.32: Front panel of PID controller implementation

$$G_c(z) = \frac{33.76z^2 - 59.98z + 26.26}{z^2 - 1} \tag{3.39}$$

The front panel of the tuned PID controller is shown in Fig. 3.33. The reference input is represented by the blue line on the plot and the actual position is represented by the red line. An input of $\pi/2\ rad$ is applied at approximately 2.6 seconds. Data collected from the deployment is exported to MATLAB and plotted in Fig. 3.34 with the upper and lower 2% settling time bound included. Although simulation results were not obtained prior to deployment of this controller, a simulated response was later generated in MATLAB and is included in Fig. 3.34 for comparative purposes. The time scale has been adjusted so that time starts the instant the reference signal is delivered. The plot shows that the actual settling time is $T_s = 1.07\ s$ which is 7% slower than the specified settling time of $T = 1\ s$. Further performance and design analysis is given in the results chapter.

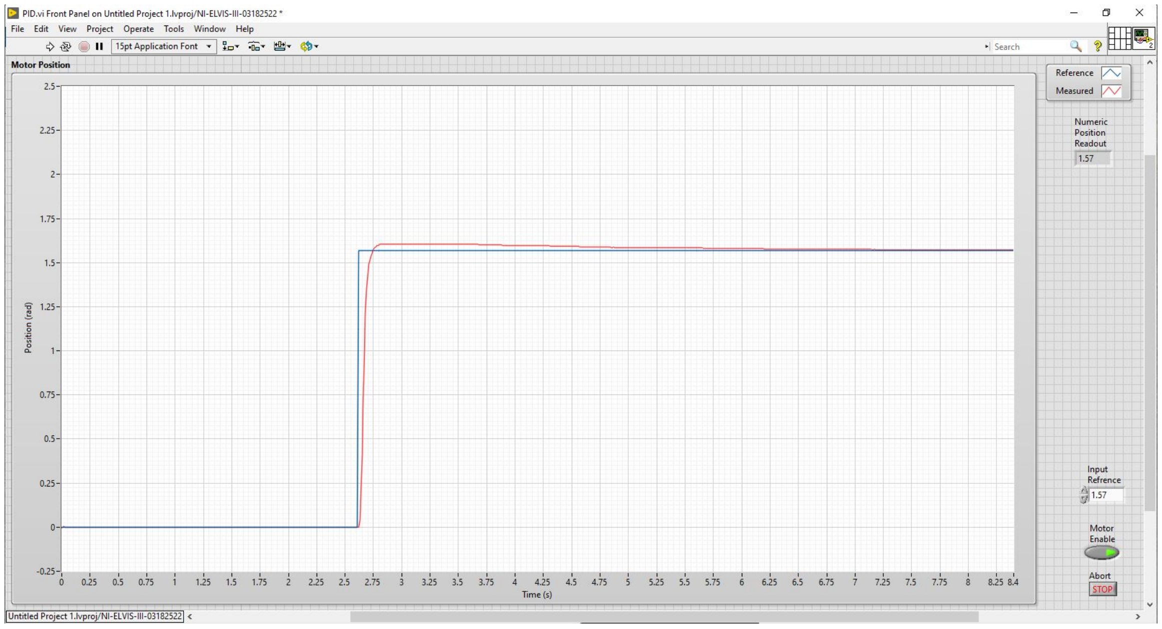

PID.vi Front Panel on Untitled Project 1.lvproj/NI-ELVIS-III-03182522 *
File Edit View Project Operate Tools Window Help
15pt Application Font
Motor Position
Reference
Measured
Numeric Position Readout
1.57
Input Refrence
1.57
Motor Enable
Abort
STOP
Position (rad)
Time (s)
Untitled Project 1.lvproj/NI-ELVIS-III-03182522

Figure 3.33: Front panel of tuned discrete-time PID controller implementation

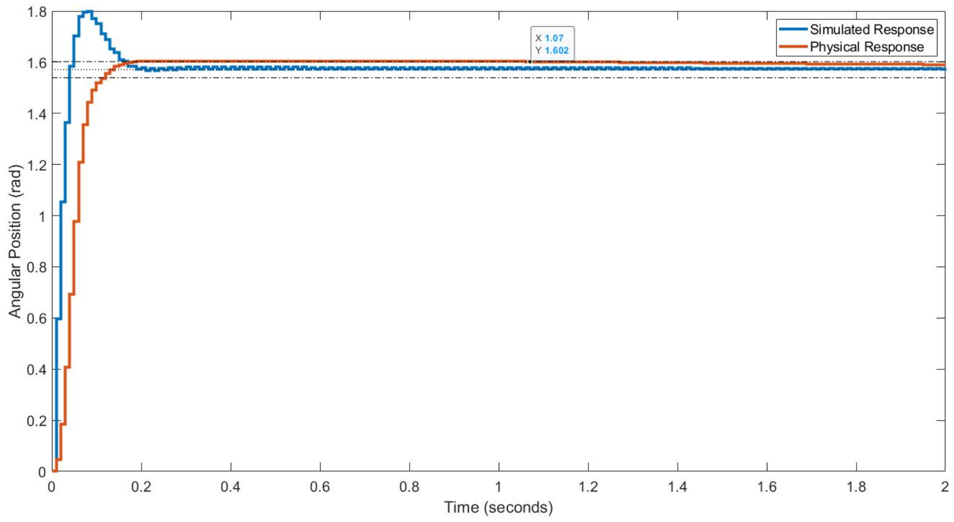


Figure 3.34: Discrete-time PID implementation

Chapter 4: Results

Having deployed all controller designs to the E/Q testbed, our focus turns to analysis. This chapter begins with a performance analysis for each controller designed under the neoclassical framework. We will then analyze the neoclassical design process using the PID controller as a standard for comparison. The chapter concludes with a discussion on the strengths and weaknesses of the E/Q testbed.

4.1 Neoclassical Performance Analysis

In this section, we will analyze the performance of all neoclassical controllers deployed to the E/Q testbed. As part of the design process, simulations were created for all the controllers designed in this work. These simulations represent the expected response of the physical system. We will therefore assess the performance of the neoclassical controllers by comparing the physical responses to the responses obtained through simulation. We will use performance characteristics such as settling time and percent overshoot to characterize performance. We will then make conclusions regarding the performance of each controller.

4.1.1 Continuous Time

Before we begin the performance analysis, we begin by briefly summarizing the continuous time neoclassical design and implementation process. Because we did not have the ability to measure all the states of the system, the continuous time neoclassical controller was based on the second order model. The second order model relied on an assumption that the motor inductance was negligible compared to motor resistance. This assumption allowed us to eliminate motor current as a state. We then used the neoclassical technique to design a controller that matched the closed-loop transfer function of the controlled system to an optimized ITAE transfer function. The resulting system was designed to have no steady-state error and a settling time of one second to an applied step input. The system was then deployed to the E/Q testbed for implementation.

In the background, the E/Q testbed converted the continuous time design to differential equations. Solutions to these differential equations were obtained through numerical approximation methods which effectively discretized the continuous time controller. The numerical approximations were then used by the E/Q testbed to generate a control signal that was delivered to the motor on the E/Q testbed. We were able to show that the implemented continuous time neoclassical controller was able to control the position of the motor with zero steady-state error and a settling time 15% slower than specified.

Figure 4.1 shows the response of the physical and simulated systems to an applied step input of magnitude $\pi/2\ rad$. Markers are provided to show settling times and peak responses. Data obtained from Fig. 4.1 is organized in Table 4.1.

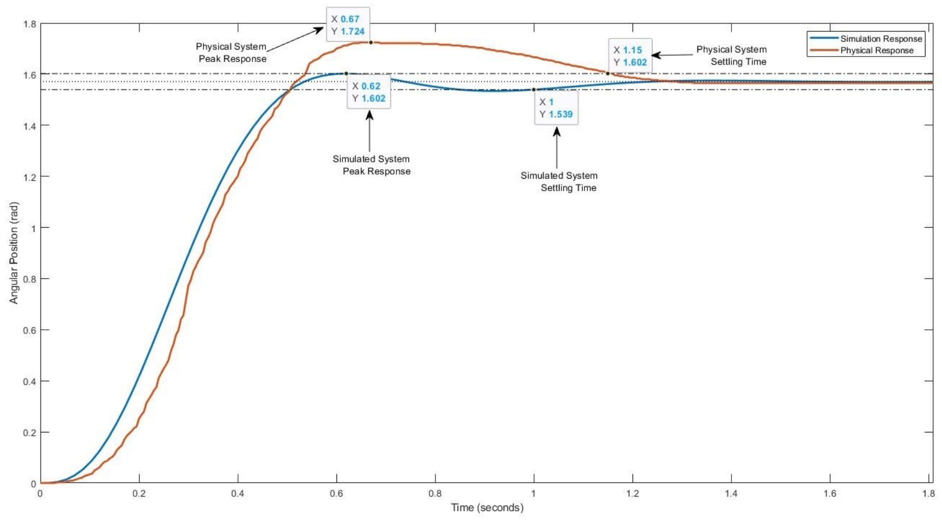


Figure 4.1: Performance analysis of continuous time neoclassical controller

Table 4.1: Continuous time neoclassical controller performance analysis

	Percent Overshoot	Settling Time
Simulated System	1.99 $\%$	1 s
Testbed System	9.75 $\%$	1.15 s
Percent Difference	389.95 $\%$	15 $\%$

The percent difference calculation shown in Table 4.1 measures the deviation between the physical response and the expected response obtained from simulation. Percent difference is calculated using (4.1). From the table, we see that the physical system has a percent overshoot of 9.75$\%$ which corresponds to a percent difference of 389.95$\%$. In other words, the percent overshoot of the physical response is 389.95$\%$ greater than our expectation. Additionally, the peak response and settling time characteristics show 7.62$\%$ and 15$\%$ differences respectively. There are several possible sources that can account for the deviation between the response obtained from the E/Q testbed and the simulation response. The first source we will discuss is the effect of variation in parameters. Next we will consider the model; then finally, the method used for programming and deploying the controller.

$$\text{percent difference} = \frac{|\text{testbed} - \text{simulation}|}{\text{simulation}} \times 100\% \tag{4.1}$$

We began the continuous time neoclassical controller design process by developing a model for the motor. We first derived the model symbolically using Newton's and Kirchhoff's laws to describe the mechanical and electrical aspects of the motor. We then substituted nominal values obtained from a data sheet provided by Quanser (Table 3.1) for the symbolic parameters. We must recognize that the values contained in any data sheet are not exact values. Rather, they are values that are bounded by some tolerance. In this case, the tolerance was not provided in the Quanser data sheet. We now consider what effect variation in parameters plays on the response of our system. To proceed we will consider the effect in a range of variations of resistance, R_m, moment of inertia, J_{eq}, and the motor constants, k_t and k_m. The second order model used for the continuous time implementation is repeated below for convenience.

$$\begin{bmatrix} \dot{\theta}_m(t) \\ \ddot{\theta}_m(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{k_t k_m}{J_{eq} R_m} \end{bmatrix} \begin{bmatrix} \theta_m(t) \\ \dot{\theta}_m(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_t}{J_{eq} R_m} \end{bmatrix} v_m(t)$$

We first consider the effect of variation in motor resistance, R_m. To facilitate this consideration, we will design and deploy two additional controllers to the E/Q testbed. The controllers are designed with an arbitrarily chosen $\pm 3\%$ ohmic deviation from nominal. The responses are compared to provide insight as to the effect of variation in the resistance parameter. A plot showing the resulting responses is shown in Fig. 4.2. Inspection of the plot reveals a high degree of variation between responses. This suggests that variation between nominal and actual motor resistance can impact the overall response. It is noted that the effect of varying moment of inertia, J_{eq}, is identical to the effect of varying motor resistance. This is because J_{eq} and R_m always appear in pairs in the second order model.

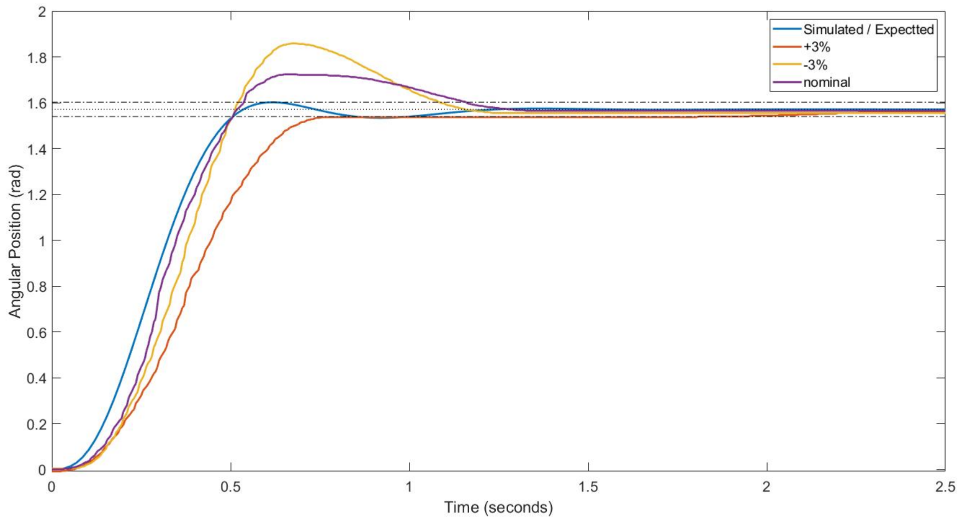


Figure 4.2: Effect of varying motor resistance by $\pm 3\%$ from nominal

Next, we consider the effect that variation of the motor constants, k_t and k_m, has on the response of the system. It should be noted that while these constants appear as unique symbols in our model, they are in fact equivalent in Table 3.1 (i.e. $k_t = k_m$). We investigate the effect of variation in this parameter by again designing and deploying two new controllers to the E/Q testbed. These controllers are designed with motor constants that have an arbitrarily chosen $\pm 3\%$ differential from nominal. A plot showing the resulting responses is shown in Fig. 4.3. We again note a significant degree of variability between responses. This again suggests that variation between nominal and actual motor constants can impact the overall response.

The parameter variation analysis shows that the response of the system can indeed be affected by variations in parameters from nominal to actual values. Next, we consider the model used for the design of the controller.

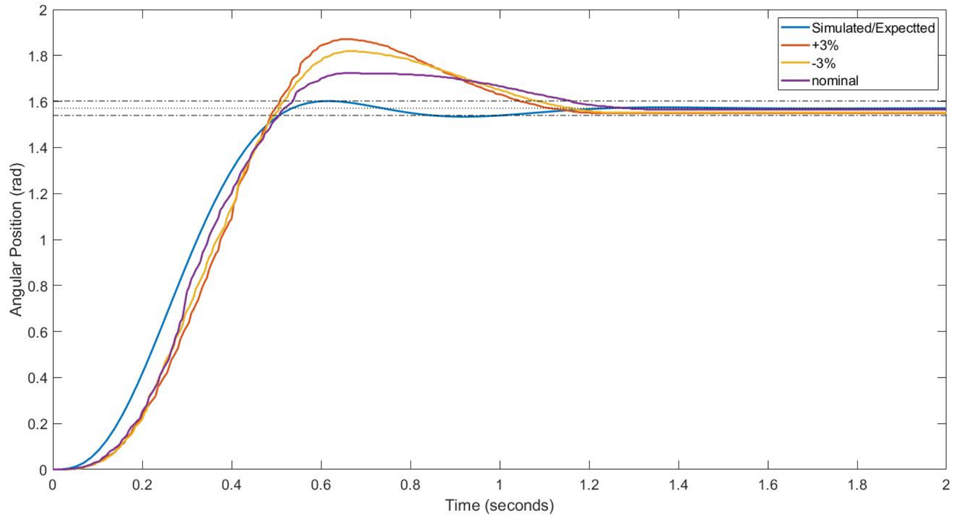


Figure 4.3: Effect of varying motor constant by $\pm 3\%$ from nominal

Another potential source that may have caused deviation between the E/Q and simulated responses is with regard to the model we used. We used the second order model that relied on an assumption which allowed us to eliminate motor current as a state. While we showed this assumption was reasonable, we effectively ignored some of the dynamics of the physical system when designing and simulating the controller. Additionally, the model of the motor was ideal in the sense that it did not consider damping (frictional forces). Quanser does not include information about damping in their documentation. It is possible that the use of the reduced order model and other un-modeled dynamics introduced undesired behavior.

The final potential source of error comes from the E/Q testbed itself. As discussed earlier, the E/Q testbed uses numerical approximations methods to implement continuous time controllers. The question is, how good were the approximations? This question will be answered later in the chapter when we analyze the E/Q testbed.

4.1.2 Discrete Time

We begin the performance analysis of the discrete time neoclassical controller by briefly summarizing the design and implementation process. The design process began by selecting a model for the system. We chose to use the second order model because we were unable to measure one of the states, motor current, included in the full third order model. Then, similar to the continuous time case, we used the neoclassical technique to design a controller which matched the transfer function of the controlled system to an optimized ITAE transfer function. The resulting system was designed to have no steady-state error and a settling time of one second to an applied step input. We determined that a sampling time of $T_s = 10\ ms$ was appropriate for our design. With this sampling time, we used the zero-order hold method to discretize the controller. The system was then verified through simulation and deployed to the E/Q testbed. The resulting system was shown to control the position of the motor with zero steady-state error and a settling time 15% slower than specification.

Figure 4.4 shows the response of the physical and simulated systems to an applied step input of magnitude $^{\pi}/_{2}\ rad$. Markers are provided to indicate settling times and peak responses. Data obtained from Fig. 4.4 is organized in Table 4.2. From the table, we see that the physical system has a percent overshoot of 3.13% which corresponds to a percent difference of 57.29%. The settling time has a percent difference of 15% greater than expected. Similar to the continuous time case, there are several factors that can contribute to the variation between the physical and simulated responses.

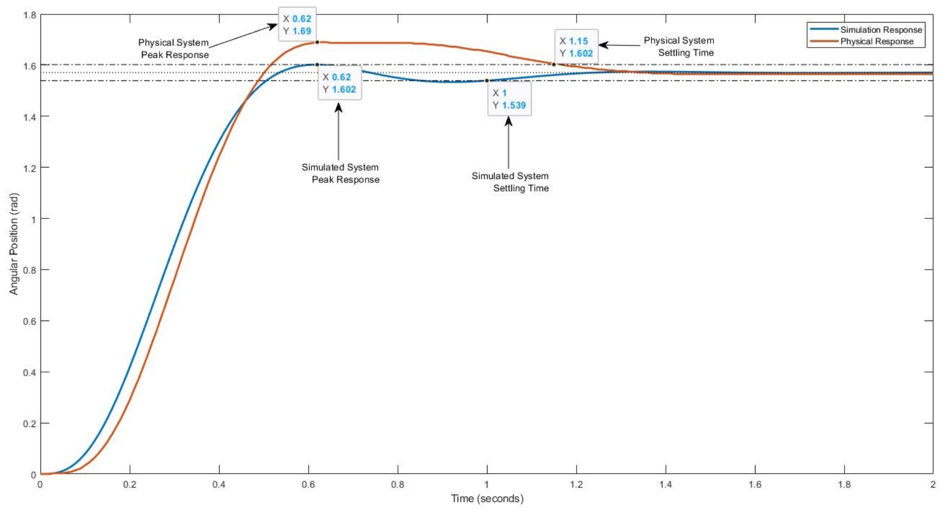


Figure 4.4: Performance analysis of discrete time neoclassical controller

Table 4.2: Discrete time neoclassical controller performance analysis

	Percent Overshoot	Settling Time
Simulated System	1.99 %	1 s
Testbed System	3.13 %	1.15 s
Percent Difference	57.29 %	15 %

When conducting the analysis for the continuous time case, we saw that the response of the system was able to be effected by variation to motor resistance, moment of inertia, and the motor constants. The discrete implementation used the same model in the design process. Further, we used the zero-order-hold method of discretization, which has the benefit of providing for what is known as exact discretization. The zero-order-hold method is sometimes referred to as exact discretization because the discrete realization is identical to the continuous time realization at every time step. Because the discrete system was designed using the same model as the continuous time system, and because we used exact discretization, it follows that the same observations made regarding variation in continuous time parameters hold for the the discrete implementation. It is therefore possible that variation between nominal and actual motor parameters contributed to the discrepancies observed between the physical and simulated systems.

Additionally, and identically to the continuous time case, we effectively ignored some of the plant's dynamics by relying on a reduced order model and ignoring the effects of damping for the design. It is, again, possible that the undesired behavior is the result of un-modeled dynamics.

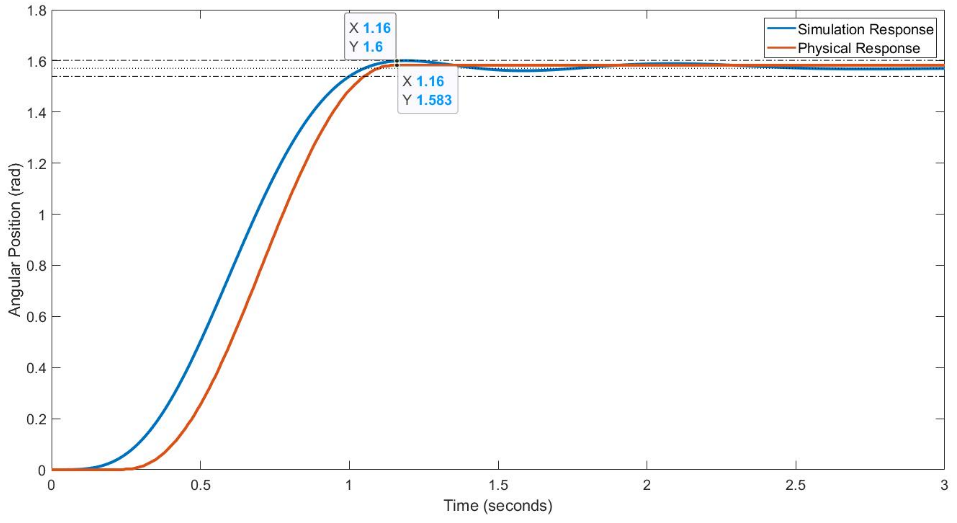
In the continuous time case, we noted that the E/Q testbed may have introduced some of the observed variation. However, this is not true for the discrete time implementation. The loop structure used in the discrete implementation did not rely on numerical approximations. Since all numerical approximation methods introduce some degree of error, we would expect the variation of the physical response from the simulated response to be less than observed in the continuous time case. The discrete system indeed showed less variation from the simulated response than the continuous time case. While both implementations resulted in the same settling time, the discrete controller had far less overshoot than the continuous time controller with percent differences of 57.29% and 389.95% respectively.

4.1.3 Discrete Time with Observer

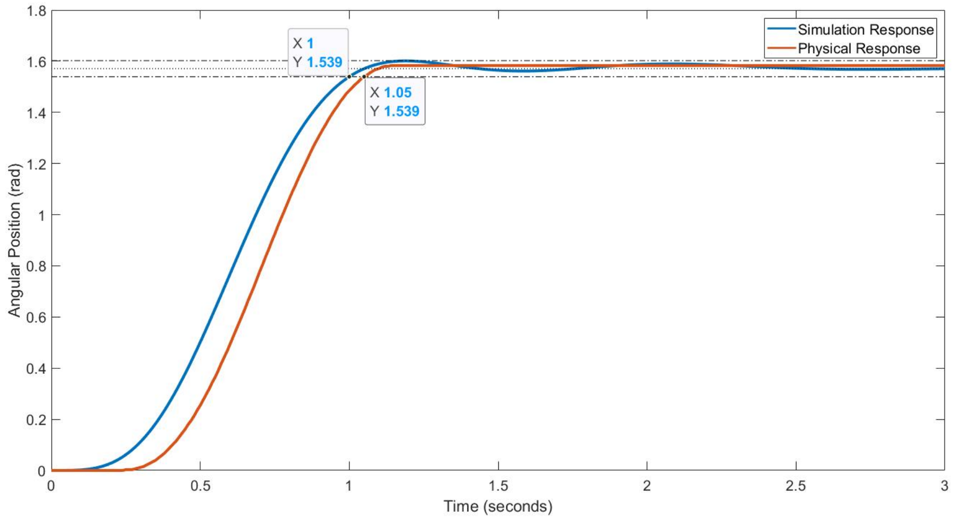
We begin the performance analysis of the of the observer based discrete time neoclassical controller by briefly summarizing the design and implementation process. We began the design process by selecting a model for our system. The previous designs required us to use the second order model because we were unable to measure one of the states of the system. This design uses an observer capable of estimating the unknown state. The observer allows us to design the controller based on the full third order model. We designed the controller using the neoclassical framework and matched the transfer function of the controlled system to an ITAE optimized transfer function. The controller was designed to have a settling time of one second and have no steady-state error to an applied step input. The observer was also designed using the neoclassical technique. Because we needed the observer to converge quickly to the actual value, we chose the observer's settling time to be 10 times faster than that of the controller. We had determined that $T_s = 10\ ms$ was an appropriate sampling time for our system. Using the sampling time, we discretized the controller and validated it through simulation. The resulting system was then deployed to the E/Q testbed where it was shown to control the position of the motor with zero steady-state and a settling time 5% slower than specification.

Figure 4.5 shows the response of the physical and simulated systems to an applied step input of magnitude $^{\pi}/_{2}\ rad$. Markers are provided to indicate settling times and peak responses. Data obtained from 4.5 is organized in Table 4.3. From the plot, we can see that the observer based discrete time neoclassical controller has a much lower degree of variation between the physical and simulated responses than was observed in the continuous and discrete time cases. From the table, we see that the physical system has a percent overshoot of 0.78% which corresponds to a percent difference of 58.06%. The settling time has a percent difference of 5% less than expected. While the response is closer to expectation than the previous controllers, it is clear from inspection of Fig. 4.5 that the trajectory of the physical response does not match the trajectory of the simulated response.

As shown previously, the difference between the physical and expected responses can be introduced due to variation between nominal and actual values of the motor parameters. Additionally, error can be introduced through un-modeled dynamics, such as damping, and the discretization process. We first attempted to discretize the controller using the zero-order hold method which, as discussed in Chapter 2, provides for an exact discretization. In this case, the zero-order hold method resulted in high observer gains that lead to instability. We instead discretized the system using the pole-zero matching method which is not an exact means of discretization.



(a) Peak response data points



(b) Settling time data points

Figure 4.5: Observer based discrete time neoclassical controller performance analysis

Table 4.3: Observer based discrete time neoclassical controller performance analysis

	Percent Overshoot	Settling Time
Simulated Response	$1.86\ \%$	$1\ s$
Physical Response	$0.78\ \%$	$1.05\ s$
Percent Difference	$58.06\ \%$	$5\ \%$

4.2 Design Analysis - Neoclassical versus PID

We now analyze the neoclassical design process. In particular, we are focused on determining if the neoclassical method simplifies the design process by eliminating the need to determine pole and zero locations for the closed-loop transfer function. To assist in this analysis, we will compare the process used to design controllers under the neoclassical framework to the process we used to design the PID controller.

The neoclassical technique strives to simplify controller design by eliminating tedious iterations and guesswork common to many design methods. To accomplish this, the neoclassical technique relies on the ability to match the transfer function of a closed-loop system to a pre-optimized transfer function. In this work, we focused on the ITAE transfer function which optimizes settling time and steady-state error. By doing so the only parameter we needed to be concerned with was the settling time of the system. Choosing a reasonable settling time for the system allowed us to obtain the form of the ITAE optimal transfer function. There was no need for us to be concerned with any other performance criteria because these characteristics were pre-determined by the ITAE optimization.

Once the desired closed-loop transfer function is known, we are tasked with creating the closed-loop system by adding feedback gains and pre-compensation around the plant. The purpose of the feedback gains is to move the poles of the closed-loop transfer function such that they match the poles of the ITAE optimal transfer function. In other words, the feedback gains allow us to match the denominator of the closed-loop transfer function to denominator of the ITAE optimal transfer function. The pre-compensator is then used to cancel the numerator in the transfer function of the plant and replace it with the numerator of the ITAE optimal transfer function.

In the continuous time case, the design process is concluded in three simple steps; 1) choose a reasonable settling time, 2) determine feedback gains, 3) define the pre-compensator. The only unknown in this process is settling time, which can be chosen with little effort. Completing these steps is said to guarantee an ITAE optimal response without needing to further tune the controller. The design process essentially ends here. It was shown that the simulated continuous time system yielded the exact response we were expecting. The deployment of the continuous time controller to the E/Q testbed showed some variation in the response which is believed to be caused by a combination of factors including; the use of the reduced order model, other un-modeled dynamics (such as damping), variation in motor parameters, and by error introduced through the approximation methods used by the E/Q testbed. The difficulty in the design process was trivial; however, the physical response failed to meet the settling time specification. While the physical response yielded zero steady-state error it took a much different trajectory than the simulated response.

The discrete case was nearly identical to continuous time case. The only additional steps needed were to select a sampling time and discretize the system. The sampling time was found with minimal effort because we knew the form of the closed-loop transfer function in advance. Using the sampling time, we then discretized the ITAE transfer function and plant. Feedback gains were used to move the discrete closed-loop poles to match the discrete ITAE pole locations. Pre-compensation was then used to match the numerator of the closed-loop system to the numerator of the ITAE transfer function. Just like the continuous time case, the only unknown parameter was the settling time. Once a reasonable settling time was determined, all other parameters were able to be found with minimal effort. The simulation of the discrete neoclassical controller showed the response of the system was precisely what we expected.

Similar to the continuous time case, the deployment to the E/Q testbed showed some variation. This variation is believed to have resulted from the use of the reduced order model, other un-modeled dynamics (such as damping), and variation in motor parameters. The design was able to control the position of the motor with zero steady-state error; however, the trajectory of the physical response did not match the simulated response. The difficulty in the design process was trivial; however, similar to the continuous time case, the physical response failed to meet the settling time specification.

The observer based neoclassical controller was designed similar to the discrete case. The design consisted of two semi-independent systems: the controller and the observer. The controller portion of the design was carried-out identically to the discrete time case. The design of the observer required a second set of poles to be placed. We again used the ITAE transfer function to determine these pole locations. We chose the settling time for the observer's ITAE transfer function to be 10 times faster than the settling time of the controller because we wanted the state estimates to converge to the true state value faster than the controller settling time. As was true in the previous two cases, once we chose a reasonable settling time all other parameters were able to be determined with ease. The simulation results confirmed that the design resulted in exactly the response we were expecting.

Implementation to the E/Q testbed revealed the need for some additional work. The discretization method used caused some instability in the response; however, this was able to be overcome by simply selecting another method for discretization. The resulting physical response showed some variation from the expected response. This variation was again attributed to parameter variances, the discretization method used in the final design, and un-modeled dynamics (such as damping). The difficulty in the design process was minimal. The system was able to control the position of the motor with zero steady-state error; however, as was the case with all neoclassical controllers, the trajectory of the physical response did not match the trajectory of the simulated response. Additionally, the observer based neoclassical controller failed to meet the settling time specification.

The PID controller has been around for many years. It is commonly used and has been studied extensively. As a result, there are a number of methods that can be used for PID controller design. Because we wanted to draw conclusions about the simplicity of the neoclassical technique, it followed that we should choose the simplest method for PID controller design. As a result, we chose to use the PID tuner toolbox in MATLAB. The PID tuner toolbox allowed us to design a controller by adjusting two slide bars. One slide controlled the speed of the response and the other controlled the robustness of the response. The step response of the PID controlled closed loop was shown in a display window. Any modification to either of the slide bars automatically updated the controller parameters and the step response in the display. This allowed us to design the PID controller with relative ease.

Since settling time was the only parameter we were required to specify when designing the neoclassical controllers, it was used as a design constraint for the PID controller. We also chose the system to be highly robust to compensate for variation between actual model parameters and the model parameters provided in the data sheet. With the settling time and robustness considerations, we were able to design a controller very quickly; however, we ran into problems during the discretization process. We found that some discretization methods resulted in a controller in the form of an improper transfer function. Because improper transfer functions are not permitted for use on the E/Q testbed, we were unable to use these methods. We found that other discretization methods resulted in system instability. Eventually, we determined that the bilinear transformation resulted in a proper controller transfer function and a stable system.

Simulation of the discretized PID controller showed promising results; however, the controller produced poor results when deployed to the E/Q testbed. It was clear on observation that the PID controller failed to come close to the settling time specification. The variation between simulated and physical responses is attributed to the discretization method, and un-modeled dynamics. We decided to improve the response by making adjustments to the controller gains. Using the original controller gains as a starting point, and having an understanding as to the effect increasing/decreasing individual gains, we manually tuned the controller. After several iterations we arrived at a controller that was 7% slower than the settling time specification. The PID was able to control motor position with zero steady-state error; however, similar to the neoclassical controllers, the physical response did not match the simulated response. While all controllers designed in this work showed promising simulation results, none yielded the physical response we were expecting.

4.3 E/Q Testbed

This section focuses on evaluating the usefulness of the E/Q testbed in validating the neoclassical design technique. Ease-of-use was a major consideration for the testbed; therefore, we begin by examining the level of difficulty required to deploy the neoclassical controllers to the E/Q testbed. The section concludes with an overall evaluation of the E/Q testbed.

Implementation of all controllers on the E/Q testbed required some understanding of LabVIEW. For continuous time controller design, the user needs to be somewhat familiar with the functions contained within the Control & Simulation Pallet. In particular, the user needs to understand how the Control & Simulation Loop structure converts the design into a physical controller. Recall that continuous time controllers must be deployed using the Control & Simulation Loop structure. This loop structure is required because LabVIEW relies on numerical approximation methods to effectively discretize continuous time controllers before deployment. Without this understanding, there is little hope the user will be able to successfully implement a continuous time controller.

Continuous time controllers can be designed once the Control & Simulation Loop structure has been configured. The design process from this point should be fairly intuitive provided the user has the background necessary to design a controller in general. The functions used for controller design are all contained within the Control & Simulation Pallet. These functions should be immediately recognizable to anyone with a general background in controls because they resemble, and are referred to by, industry standard symbols and names. For example, to include a summing junction in the design, the user would open the Control & Simulation Pallet and locate the function that resembles the block diagram representation of a summing junction and is titled "Summation." Because LabVIEW uses these functions in a graphical programming environment (see section 2.1), continuous time controller design can be accomplished by those with little to no LabVIEW experience.

While the deployment of the continuous time neoclassical controller was able to control the position of the motor, we did note some error in the response. This error was likely the result of four factors; variation in motor parameters, the reduced order model used for the design, un-modeled dynamics, and the numerical approximation method we chose. We had little control over our choice in model; however, we did have some control over the numerical approximation method used by the Control & Simulation Loop structure. While we are required to use numerical approximations for continuous time controller deployment, we are able to choose the method and step size. In particular, we are free to use any of the fixed step size approximation techniques which are Runge-Kutta orders 1-4. As discussed in Chapter 3, choosing the numerical approximation method, and step size, required a test-and-adjust approach. It is noted that having some background in numerical analysis can greatly assist the user in this process.

A plot comparing the continuous and discrete time neoclassical controller is shown in Fig 4.6. Recall that the discrete neoclassical controller was designed using the same model used for the continuous time controller. Because we used exact discretization (zero-order hold) for the discrete implementation, we can regard most of the variation between the continuous and discrete responses as being a result of the numerical approximation methods used by the Control & Simulation Loop structure. We can see from Fig. 4.6 that the responses are very similar; therefore, the error caused by the numerical approximation method is regarded as negligible. It should be noted that several numerical approximations methods were attempted in this work. We found through trial and error that the third order Runge-Kutta method with a step size of $5\ ms$ provided the response that exhibited the least error.

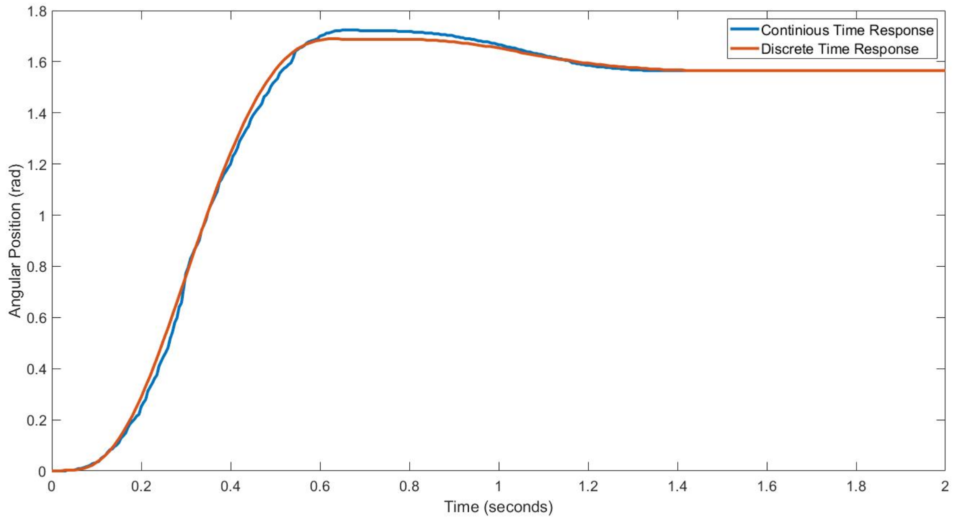


Figure 4.6: Comparison of continuous time neoclassical response to discrete time neoclassical response

While the Control & Simulation Loop structure can be confusing; National Instruments provides a help file to assist. The remainder of the controller was designed with little effort because of the intuitive nature of LabVIEW's graphical programming environment. The design and implementation processes were accomplished with little effort; however, we did not achieve the response we expecting from the controller.

The discrete time neoclassical controller was designed similar to the neoclassical controller. The main difference is the loop structure used for discrete time implementation. Because there was no need for LabVIEW to discretize a discrete controller, we were not confined to using the Control & Simulation Loop structure. Instead, we used the Timed Loop Structure which does not rely on numerical approximation methods. Rather, the loop iterates instruction at a set iteration period. The iteration period is easy to define because it is equivalent to sample time. The loop itself behaves similar to a zero-order hold because it maintains its output between iterations.

Once the Timed Loop structure has been configured with the appropriate iteration period, the controller design process proceeds similarly to the continuous time controller design. The functions used to create a controller are located on the Control & Simulation and Numerical pallets. The major difference between creating continuous and discrete controllers is the placement of the functions.

In the continuous time case, LabVIEW converted the design into differential equations. As a result, the placement of the functions did not matter. This is not the case for the discrete time design. The Timed Loop structure does not convert the controller into differential equations. Instead, execution of instructions occurs primarily from left to right. Therefore, we need to consider the order in which information is processed and place the functions in appropriate locations. This can be somewhat confusing to those unaware of how a Timed Loop structure operates; however, National Instruments provides a help file to assist.

The discrete time neoclassical controller was able to be designed and deployed with minimal effort. Similar to the continuous time case, the design and implementation process was accomplished with little effort; however, we did not achieve the response we expecting from the controller.

The observer based neoclassical controller appeared to be far more complicated than its non-observer based counterpart; however, the additional components did not add much complexity. Much of the controller was designed identically to the discrete time controller. There were, however, two major differences; the observer VI and some additional attention required in placement of the VIs. The observer VI is fairly intuitive provided the user has the background necessary to design an observer in the first place. Placement of the VIs; however, becomes a bit more complex when using an observer. The observer estimates the states of the system at the next time step. As a result, we need to store the state estimation information for use in the next iteration. We accomplished this by using a feedback node.

The observer based neoclassical controller was very similar to the discrete case. There was a bit more complexity required to incorporate the observer; however, the additional complexity was not considered to be significant. The physical output of the observer based neoclassical controller was shown to have some error when compared to the simulated response. The error that was observed was attributed to the method used to discretize the controller, un-modeled dynamics, and variation in parameters. As was the case for all neoclassical controllers, the design and implementation process was accomplished with little effort; however, we did not achieve the response we expecting from the controller.

While the E/Q testbed is indeed easy to use; none of the responses of the controllers deployed to the testbed matched their simulated responses. We, therefore, cannot conclude that the E/Q testbed is useful for validating experimental controllers. As a result, we can draw no conclusions as to the validity of the neoclassical design technique.

Chapter 5: Conclusion

The chapter serves as the conclusion to this thesis. We begin by summarizing the work completed. Next, conclusions are made regarding the usefulness of the E/Q testbed and the effectiveness of the neoclassical design technique. This chapter closes with a discussion of future work for the E/Q testbed and neoclassical technique.

5.1 Summary of Thesis

Control systems are used in many applications. Some of these applications can be rather complex and may include expensive hardware. Control systems themselves are constantly evolving and new methods for controller design are continually being developed. Before controllers designed using new, and untested, design methodologies are deployed on a complex and expensive systems; they should first be validated. The main objective of this thesis was to develop a testbed for the purposes of validating experimental controllers. A secondary objective was to use the testbed to validate three controllers designed using the neoclassical design technique.

The testbed was designed using the National Instruments ELVIS III and Quanser Controls Board. The ELVIS III contains a microprocessor and FPGA which facilitate controller implementation. The Quanser Controls board consists of a motor and two encoders which serves as the system we wish to control. The combination of the ELVIS III and Quanser Controls board comprises the testbed which we have called the E/Q testbed. Controllers are designed using LabVIEW software from National Instruments. LabVIEW uses a graphical programming environment which promotes ease-of use. A controller can be deployed to the E/Q testbed once it has been designed in LabVIEW.

We used the neoclassical design technique to evaluate the usefulness of the testbed. The neoclassical technique was chosen because up until this point it had never been used to control physical hardware. The neoclassical design technique claimed to give rise to an efficient design process by allowing the designer to match the transfer function of a controlled system to a transfer function that has been pre-optimized to elicit a particular response. To accomplish this, the neoclassical technique uses elements from modern and classical approaches to controller design. The modern contribution comes in the form of state variable feedback which matches the denominator of the controlled system's transfer function to the denominator of pre-optimized transfer function by moving the poles of the system. The classical contribution to the neoclassical controller is the pre-compensator which matches the numerator of the controlled system's transfer function to the numerator of the pre-optimized transfer function by placing a gain in cascade with the plant.

Three controllers were designed under the neoclassical framework to evaluate the usefulness of the E/Q testbed. A PID controller was designed for comparative purposes and used to evaluate the efficiency of the neoclassical design process. We used the discrete PID as a comparative standard because it is well known and understood.

All controllers were designed to control the angular position of the motor located on the E/Q testbed. To be thorough in evaluating the E/Q testbed, and the neoclassical method, controllers were designed in both continuous and discrete time. Both continuous and discrete time controllers were designed using a reduced order model of the plant. It was necessary to reduce the order of the model because one of the states, motor current, was not available for measurement. The third neoclassical controller implemented a state observer which was used to measure the unknown state. The observer allowed us to design the controller using the full order model of the plant. All controllers were first designed using MATLAB. Simulation was then used to verify each controller design. Once verified, the controllers were deployed to the E/Q testbed where their responses were recorded.

We considered the physical and the simulated responses when evaluating the E/Q testbed and the neoclassical design technique. The simulated response represented the response we expected from the physical system. We considered any deviation of the physical response from the simulated response to be an error. After all controllers were deployed, we analyzed the performance of each controller. This analysis was based on the error of the responses and performance characteristics such as peak response, percent overshoot, settling time, and steady-state error. Through this process we were able to make determinations regarding the usefulness of the E/Q testbed in evaluating the neoclassical controller and the neoclassical technique as a controller design framework. These determinations will be discussed in the next section.

5.2 Conclusions

All controllers designed in this work were able to be deployed to the E/Q testbed. In each case, the controllers were required to control the angular position of the motor shaft located on the E/Q testbed. The systems were given a reference of $^{\pi}/_{2}\ rad$, and were expected to rotate the shaft of the motor to angular position of $^{\pi}/_{2}\ rad$ from the origin. In every case the controllers were able to complete this task with zero steady-state error; however, none of the controllers physical responses were able to match the trajectory of their simulated responses.

Programming the controllers using LabVIEW proved to be efficient. The graphical programming environment ensured that most of the programming could be completed by persons having little experience using LabVIEW. The control & simulation loop structure as well as the placement of the functions in the timed loop structure were shown to require a bit more attention to configure. It was noted that without a prior knowledge of how the Control & Simulation Loop structure functions, and how the Timed Loop structure executes instruction that there was little hope of being able to successfully deploy a controller to the E/Q testbed.

The continuous time controller was shown to have some error. This error was attributed to a combination of parameter variation, un-modeled dynamics, the use of a reduced order model, and error introduced by the E/Q testbed. Before the controller is deployed, the E/Q testbed first converts the design to differential equations. Solutions to the differential equations are approximated using numerical approximation methods. Because numerical approximation methods rely on time steps; this process effectively discretized the controller. The resulting controller is then deployed to the E/Q testbed. Because the implementation is based on numerical approximations, the response of any continuous time controller deployed to the E/Q testbed will always exhibit some error. The amount of error can be minimized through selection of numerical method and step size; however, it can never be zero. While the controller's response did not exactly match the simulated response, it was able to successfully control the position of the motor.

The discrete time and observer based discrete time controllers proved to be less time consuming to implement than the continuous time controller. The main reason for this was due to fact that the discrete implementations did not rely on numerical approximation methods. As a result, we had no need to make adjustments to the numerical approximation method, and step size, in an effort to reduce approximation error. We did; however, need to be mindful of the location in which we placed VIs within the timed loop. The Timed Loop primarily executes instruction from left to right within the loop structure. When placing VIs, we needed to make certain instructions were processed in a logical order to avoid unexpected behavior.

The performance of the discrete time and the observer based discrete time neoclassical controllers were evaluated by comparing the physical responses to the simulated responses. Both controllers were shown to have zero steady-state error to the applied step input. Some error was present in the responses of both controllers. The error in the discrete time controller was attributed to the use of a reduced order model in the controller design, un-modeled dynamics, and from variation in motor parameters. The error in the observer based discrete time neoclassical controller was attributed to the method used for controller discretization, un-modeled dynamics, and from parameter variation.

Design analysis was made by comparing the design process of the neoclassical technique to that of the PID. The neoclassical process was shown to be highly efficient because there was no need to determine where the poles and zeros of the closed-loop system should be located. Once the designer selected a desired settling time all design parameters can be determined with little effort and an optimized response is guaranteed. Because it promotes ease-of-design; we chose to use the PID tuner toolbox in MATLAB to design the controller. The initial design was very simple and took a minimal amount of time. We then manually tuned the gains of the PID until we obtained a response that was closer to the design specifications. All controllers deployed to the testbed were able to control the position of the motor with zero steady state error; however, none of the controllers physical responses matched their simulated responses. We therefore drew no conclusions regarding the claim that the neoclassical design technique provides an improvement over other design methods such as the PID.

The final conclusion is that the E/Q testbed is not particularly useful for the validation of experimental control techniques since none of the controllers physical responses (including the PID) matched their simulated responses. It is noted that both the ELVIS III and Quanser Controls Board are intended to be used primarily as instructional aids; to which they are well suited.

5.3 Future Work

Future work for the testbed includes configuring the ELVIS III to be used on other benchmark systems such as magnetic levitation or the mass-spring-damper systems. Additional the implementation of a reduced-order observer should be explores in LabVIEW. Currently, LabVIEW does not have a function capable of reduced-order estimation. Development of a reduced-order estimator VI would be useful for higher order systems.

Future work for the neoclassical control method also includes the implementation of a reduced-order observer. Currently, the implementation of a reduced-order observer using the neoclassical technique has never been attempted. Future neoclassical development should also consider multiple input multiple output (MIMO) systems. Use of inputs such as a ramp, or a parabola, should also be explored. Current neoclassical research has only considered a step input.

Bibliography

[1] Q. Inc., *Lab Manual: Control Systems Design and Analysis*, 2019.

[2] IEEE, “Control and control history,” Available at http://old.ieeecss.org/publications/tac/control-control-history (2020/08/20).

[3] R. Dorf and R. Bishop, *Modern Control Systems*, 13th ed. Hoboken NJ: Pearson, 2017.

[4] K. Holterman, “A neoclassical approach: More efficient controller design,” Master’s thesis, Marquette University, Milwaukee, WI, 2008.

[5] C.-T. Chen, *Linear System Theory and Design*, 4th ed. New York, NY: Oxford University Press, 2012.

[6] W. A. Baker, “Observer incorporated neoclassical controller design: A discrete perspective,” Master’s thesis, Marquette University, Milwaukee, WI, 2010.

[7] Magnetic levitation. Acumen Labware. [Online]. Available: http://acumenlabware.com/electronics-training-kits/magnetic-leviation-experiment.html (2021/4/2)

[8] Electrical test bench and motor test bench. JM Test Systems. [Online]. Available: https://jmtest.com/electrical-test-bench/#functions (2019/10/15)

[9] Ni elvis iii manual. National Instruments. [Online]. Available: https://www.ni.com/documentation/en/ni-elvis-iii/1.0/manual/manual-overview/ (2019/08)

[10] Ni measurements live. National Instruments. [Online]. Available: https://measurementslive.ni.com// (2020/09)

[11] N. Instruments, “Ni elvis iii,” Available at https://www.ni.com/en-us/support/model.ni-elvis-iii.html (2020/09).

[12] Quanser, “Quanser controls board,” Available at https://www.quanser.com/products/quanser-controls-board/#productdetails (2020/09).

[13] C. Alexander and M. Sadiku, *Fundamentals of Electric Circuits*, 5th ed. New York, NY: McGraw-Hill, 2013.

[14] D. Graham and R. C. Lathrop, “The synthesis of ”optimum” transient response: Criteria and standard forms,” *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, vol. 72, no. 5, pp. 273–288, 1953.

[15] N. Instruments, *LabVIEW Control Design User Manual*, 2009.

[16] M. S. Fadali and A. Visioli, *Digital Control Engineering Analysis and Design*, 3rd ed. Cambridge, MA: Academic Press, 2020.

Appendix A: Setting up the Testbed

This appendix provides information required to assemble the E/Q testbed and the software that will be needed to communicate with, and program, the testbed. The steps can be followed in any order.

1. Remove any previously installed top board from the ELVIS III and install the Quanser Controls board.
2. Verify LabVIEW 2019 SP1 is installed on the host computer †
3. Install the ELVIS III software bundle ††

† It may be possible to use other versions of LabVIEW. The work presented in this thesis used LabVIEW 2019 SP1. No other versions of LabVIEW were considered in this work.

†† The ELVIS III software bundle contains various software and drivers that are required to communicate with, and program, ELVIS III. The software bundle can be installed using the NI Package Manager program. Alternatively, the software bundle can be downloaded from National Instruments at www.ni.com/en-us/support/downloads/software-products/download.ni-elvis-iii-software-bundle.html#333217

Appendix B: MATLAB Code

```
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Second Order - NeoClassical - Continious Time  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 close all; clear all; clc; format compact

% Motor Constants
Rm = 8.4; Kt = 0.042; Km = 0.042; Jm = 4e-6;
% Load Hub Constants
Mh = 0.0106; Rh = 0.0111; Jh = 0.5*Mh*Rh^2;
% Disk Constants
Md = 0.053; Rd = 0.0248; Jd = 0.5*Md*Rd^2;
% EOM Constants
 Jeq = Jm+Jh+Jd;

% System Equations and Integrator
A = [0 1; 0 -(Kt*Km)/(Rm*Jeq)];
B = [0 Kt/(Rm*Jeq)]';
C = [1 0];
D = [0];
 Gsys_ss = ss(A,B,C,D);  % System state space
 Gsys_tf = tf(Gsys_ss);  % System transfer function
 int_tf = tf(1,[1 0]);   % Integrator transfer function
 int_ss = ss(int_tf);    % Integrator state space

% Define Natural Frequency
 Tdesired = 1;           % Desired settling time
 Tnorm = 7.54;           % Norm. 3rd order ITAE settling time
Wn = Tnorm/Tdesired;     % Natural frequency

% Define Desired ITAE
 T_itae_tf = tf(Wn^3, [1 1.75*Wn 2.15*Wn^2 Wn^3]); % TF
 T_itae_ss = ss(T_itae_tf);                        % SS

% Define Forward Path - Gfp(s)
 Gfp_ss = minreal(T_itae_ss / (1-T_itae_ss));    % w int
 Gfp_noInt_ss = minreal(Gfp_ss/int_ss);     % int removed
 Gfp_noInt_tf = tf(Gfp_noInt_ss);

% Find Pole Locations and Feedback Gain
 poles = eig(Gfp_noInt_tf);      % Pole location
K = place(A,B,poles);            % Gain matrix

% Define System with Stste Variable Feedback
 Gsvfbk_ss = ss(A-B*K,B,C,D);

% Define Pre-Compensation
PC = tf(cell2mat(T_itae_tf.num),cell2mat(Gsys_tf.num));
```

```

% Controlled System
sys_fp = minreal(Gsvfbk_ss*int_ss*ss(PC));  % Fwd path
CT = feedback(sys_fp,1);                    % Final system

% Plot
mag = pi/2; % Magnitude of Step Input
opt = stepDataOptions('StepAmplitude',mag);
figure(1)
step(CT,opt);
title('Simulation of Neoclassical Motor Position Control in Continious
    Time')
ylabel('Angular Position (rad)')

% Display Gain and PC (PC includes int)
Gain = K
Pre_Compensator_with_Integrator = tf(minreal(PC*int_ss))

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Second Order - NeoClassical - Discrete Time   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc; format compact

% Motor Constants
Rm = 8.4; Kt = 0.042; Km = 0.042; Jm = 4e-6;
% Load Hub Constants
Mh = 0.0106; Rh = 0.0111; Jh = 0.5*Mh*Rh^2;
% Disk Constants
Md = 0.053; Rd = 0.0248; Jd = 0.5*Md*Rd^2;
% EOM Constants
Jeq = Jm+Jh+Jd;

% System Equations, Integrator
A = [0 1; 0 -(Kt*Km)/(Rm*Jeq)];
B = [0 Kt/(Rm*Jeq)]';
C = [1 0];
D = [0];
int_tf_ct = tf(1,[1 0]);    % Integrator transfer function

% Discrete System, Sampling Period and Discrete Integrator
T = 0.01;                               % Sampling period
[Ad,Bd,Cd,Dd] = c2dm(A,B,C,D,T,'zoh');  % Discretization ZOH
Gsys_ss = ss(Ad,Bd,Cd,Dd,T);            % System state space
Gsys_tf = tf(Gsys_ss);                  % system TF
int_tf_dt = c2d(int_tf_ct,T,'zoh');     % Discrete int TF
int_ss_dt = ss(int_tf_dt);              % Discrete int SS

% Define Natural Frequency
Tdesired = 1;           % Desired settling time
Tnorm = 7.54;           % Norm. 3rd order ITAE settling time
Wn = Tnorm/Tdesired;    % Natural frequency

% Define Desired ITAE
```

```
T_itae_tf_ct = tf(Wn^3, [1 1.75*Wn 2.15*Wn^2 Wn^3]); % CT TF
T_itae_tf_dt = c2d(T_itae_tf_ct,T,'zoh');            % DT TF
T_itae_ss_dt = ss(T_itae_tf_dt);                      % DT SS

% Define Forward Path - Gfp(s)
Gfp_ss = minreal(T_itae_ss_dt / (1-T_itae_ss_dt));  % w/ int
Gfp_noInt_ss = minreal(Gfp_ss/int_ss_dt);       % int removed
Gfp_noInt_tf = tf(Gfp_noInt_ss);

% Find Pole Locations and Feedback Gain
poles = eig(Gfp_noInt_tf);      % Pole location
K = place(Ad,Bd,poles);         % Gain matrix

% Define System with Stste Variable Feedback
Gsvfbk_ss = ss(Ad-Bd*K,Bd,Cd,Dd,T);

% Define Pre-Compensation
PC = tf(cell2mat(T_itae_tf_dt.num),cell2mat(Gsys_tf.num),T);

% Controlled System
sys_fp = minreal(Gsvfbk_ss*int_ss_dt*ss(PC)/T); % Fwd path
DT = feedback(sys_fp,1);        % Final system

% Plot
mag = pi/2;     % Magnitude of Step Input
opt = stepDataOptions('StepAmplitude',mag);
figure(1)
step(DT,opt)
title('Simulation of Neoclassical Motor Position Control in Discrete
    Time')
ylabel('Angular Position (rad)')

% Display Gain and PC (PC includes int)
Gain = K
Pre_Compensator_with_Integrator=tf(minreal(PC*int_ss_dt/T))

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Third Order - NeoClassical - Discrete Time w/ observer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc; format compact

%%%%%%%%%%%%%%%%                CONTROLLER DESIGN
% Motor Constants
Rm = 8.4; Kt = 0.042; Km = 0.042; Jm = 4e-6; Lm = 1.16e-3;
% Load Hub Constants
Mh = 0.0106; Rh = 0.0111; Jh = 0.5*Mh*Rh^2;
% Disk Constants
Md = 0.053; Rd = 0.0248; Jd = 0.5*Md*Rd^2;
% EOM Constants
Jeq = Jm+Jh+Jd;
% Sample time
T = 0.01;

```

```
% System Equations
A = [0 1 0; 0 0 Kt/Jeq; 0 -Km/Lm -Rm/Lm];
B = [0; 0; 1/Lm ];
C = [1 0 0];
D = [0];

[Ad,Bd,Cd,Dd] = c2dm(A,B,C,D,T,'matched')

[Gp_z_num, Gp_z_den] = ss2tf(Ad,Bd,Cd,Dd);
Gp_z = zpk(tf(Gp_z_num, Gp_z_den,T));

% Define Natural Frequency
Tdesired_c = 1;        % Desired settling time
Tnorm4 = 4.51;         % Norm. 4th order ITAE settling time
Wn_c = Tnorm4/Tdesired_c;   % Natural frequency - controller

% Define Desired ITAE
s = tf('s');
z = tf('z',T);
itae4_ct = Wn_c^4 / (s^4 + 2.1*s^3*Wn_c + 3.4*s^2*Wn_c^2 + 2.7*s*Wn_c^3
    + Wn_c^4);
itae4_dt = c2d(itae4_ct,T,'zoh');
itae4_dt_ss = ss(itae4_dt);
Gfp_dt_ss = minreal( itae4_dt_ss / (1-itae4_dt_ss) );
Gfp_dt = minreal((z-1)*tf(Gfp_dt_ss)/T);  % int removed

% find pole locations and determine feedback gains
pols_c  = eig(Gfp_dt);
K = place(Ad,Bd,pols_c)

% Define precompensation
num_pc = cell2mat(Gfp_dt.num);      % PC numerator
den_pc = Gp_z_num;                  % PC denominator
pc = tf(num_pc,den_pc,T);           % PC
PC = minreal(ss(pc) * T / (z-1));   % PC w/ integrator

Gsvfbk = ss(Ad-Bd*K,Bd,Cd,Dd,T);    % sys w/ state fdbk
Gpc = minreal(PC*Gsvfbk);           % foward path
Gsys = feedback(Gpc,1);             % closed loop system


%%%%%%%%%%%%%%%%                 OBSERVER DESIGN
% Define Natural Frequency
Tdesired_o = .1;            % Desired settling time
Tnorm2 = 7.54;              % Normalized 3rd order ITAE settling time
Wn_o = Tnorm2/Tdesired_o;   % Natural frequency - observer

% Define desired ITAE transfer and observer gain
itae3_ct = Wn_o^3 / (s^3 + 1.75*s^2*Wn_o + 2.15*s*Wn_o^2 + Wn_o^3);
itae3_dt = c2d(itae3_ct,T,'zoh');
pols_o  = eig(itae3_dt);
L = place(Ad',Cd',pols_o)'

% rebuild system with observer
```

```
AA = [Ad-Bd*K Bd*K; zeros(size(Ad)) Ad-L*Cd];
BB = [Bd; zeros(size(Bd))];
CC = [Cd zeros(size(Cd))];
Gobs_ol = minreal(PC * ss(AA,BB,CC,D,T));
OBS = feedback(Gobs_ol,1);

mag = pi/2;       % Magnitude of Step Input
opt = stepDataOptions('StepAmplitude',mag);
figure(1)
step(OBS,opt)

preComp = tf(PC)     % display precompensation
```